Dusza–Varga

A BASIC nyelvű programozás ábécéje

MŰSZAKI KÖNYVKIADÓ

Dusza Árpád–Varga Antal

A BASIC nyelvű programozás ábécéje

Dusza Árpád–Varga Antal

A BASIC nyelvű programozás ábécéje

MŰSZAKI KÖNYVKIADÓ, BUDAPEST, 1985

Dusza Árpád–Varga Antal: A BASIC NYELV c. tankönyv változatlan kiadása

Lektorok:

Garádi János
dr. Ratkó István

© Dusza Árpád–Varga Antal

ISBN 963 10 6600-2

ETO: 519.682. BASIC
681.3.06 BASIC

Felelős szerkesztő: Tarr Kálmánné

A számítógépek kialakulása, a számítástechnika fejlődése

A számítástechnika születését és fejlődését különböző nézőpontból közelíthetjük meg. Tisztelettel adózva elődeinknek, felsorolhatnánk jeles tudósoknak, matematikusoknak a számítások meggyorsítására, automatizálására tett kísérleteit és eredményeit. Ez azonban inkább egy tudománytörténeti munka feladata lenne. Így ezekre az izgalmas előzményekre most nem térünk ki.

A próbálkozások után az első igazi „számítógépet" a 19. sz. közepén, 1822-ben korát messze megelőzve, Charles Babbage a cambridge-i egyetem tanára tervezte. A számítógép mechanikus működésű volt, a számok reprezentálására fogaskerekek bizonyos fordulatai szolgáltak.

Boole 1847-ben dolgozta ki a kétállapotú kifejezések algebráját – részletesebben lásd később –, hangoztatva, hogy felhasználásával logikai problémákat lehetne automatizálva elemezni és megoldani. A nagyszerű gondolat gyakorlati megvalósítása azonban csaknem 100 évet váratott magára. Ekkor még anyagi megvalósítás nem tudta követni a szellemi terméket.

A 20. század társadalmi és gazdasági tekintetben egyaránt hallatlan nagy fejlődést hozott. Mindez éreztette hatását a műszaki és tudományos életben, így a számítástechnika kifejlődésében is. 1911-ben alakult meg az első számítógépeket felhasználó társaság, a CTR (Computer Tabulator Recording), amely 1924-ben alakult át a világ számítástechnikai iparában a mai napig is vezető szerepet játszó IBM-mé (International Business Machines).

1939-ben, Atanasoff tervei alapján, építettek egy gépet elektroncsövek felhasználásával, amely többismeretlenes egyenletrendszerek megoldására volt alkalmas.

1941-ben Zuse irányítása mellett készül el az első elektromechanikus működésű, de általános jellegű feladatok megoldásához is használható számítógép, a Z3.

Az IBM is kialakított egy ilyen eszközt 1944-ben, amely a Harward egyetemen 1950-ig dolgozott. Hagyományos elemekből készült, és 6 s alatt tudott egy szorzást elvégezni.

1945-ben, a világon először, a magyar származású, világhírű matematikus, Neumann János tervei alapján kezdik el az USA-ban egy belső programvezérlésű számítógép kifejlesztését. Igaz, hogy csak 1952-ben készült el, EDSAC néven, de a „Neumann elv" meghatározóan fontos szerepet kapott a számítástechnika fejlődésében.

1946-ban készült el szintén az USA-ban az első elektronikus működésű, de még külső vezérlésű számítógép, amely ugyan gigantikus méretű, de az adott körülményekhez képest gyors és megbízható volt.

A számítógépek sorozatgyártását 1951-ben kezdték el az USA-ban, ennek a „gépcsaládnak" az UNIVAC nevet adták. A következő nagy ugrás a gépek korszerűsítésében a

félvezetők elterjedése. Alkalmazásával a méret és a teljesítményfelvétel csökkenése mellett a gépek üzembiztonsága és műveleti sebessége nőtt meg. Ezeknek az ún. második generációs gépeknek a fő alkotó része a tranzisztorokból épített logikai áramkör volt.

A harmadik generációhoz tartozó gépek integrált áramköri elemeket tartalmaznak (60-as évek). Ezek az egységek minimális helyfoglalással összetett logikai és aritmetikai feladatok ellátására alkalmasak. Így a fejlődés iránya a komplex számítástechnikai munkák megoldására alkalmas, nagy kapacitású óriásgépek mellett egyre inkább a felhasználóval közvetlen kapcsolatot teremtő mikrogépek fejlesztése felé is mutat. Ezeknek a személyi számítógépeknek a felhasználási területe rendkívül széles, népszerűségük a hozzáférhetőséget, árat, egyszerűséget tekintve érthető. Központi feldolgozó egységük – Central Processor Unit, CPU – egy mikroprocesszor. Ez egy olyan eszköz, amely több logikai áramkört tud egy időben helyettesíteni a legáltalánosabb feladatok végrehajtásában: elektronikus háztartási eszközök, számológépek, számítógépek stb. Az első mikroprocesszort 1971-ben készítették el. Alkalmazásával és széleskörű elterjedésével a fejlődés a kívülállók számára már szinte követhetetlenül felgyorsult. A tankönyv nagyságú személyi számítógépek használata, amelyek ráadásul nagyon könnyen elsajátítható és jól használható nyelven „értenek" (BASIC), valóban bűvészkedésnek tűnhet a laikusoknak! Hogy nem az, azt feltehetően a könyvünk tanulmányozása után mindenki beláthatja.

A számítógépek működése és felépítése

Ahhoz, hogy egy számítógépet „jó ismerősként", optimálisan kihasználva működtessünk, nem elég, ha csak értve a közös nyelvet, azon „utasítgatjuk" különböző feladatok végrehajtására, valamilyen szinten a „lelkébe" is be kell tekintetni. Ezért ismerkedjünk meg általánosságban a felépítésével, működésének alapvető logikájával és a hozzátartozó jelölésrendszerrel.

Talán még ma is sokan vannak, akikben valamilyen irányban irreális kép alakult ki a számítógépekről. Vagy leegyszerűsítve csak egy nagyon gyorsan számoló automatának képzelik, vagy pedig egy fel nem fogható, teljesen elvont elvek alapján működő eszköznek. Az ugyan tény, hogy az alapműveletek elvégzésében hihetetlenül gyorsak. A „hőskorban", az elektroncsöves gépek 1000–5000 műveletet, a tranzisztoros elemek felhasználásával 10 000–50 000 műveletet, az integrált áramköri egységekből épített gépek pedig 100 000-nél is több műveletet képesek elvégezni 1 s alatt. De ez nem minden! Többek között persze a nagy műveleti sebesség az, ami olyan feladatok elvégzésére is alkalmassá teszi a számítógépet, amelyek a gyakorlatban előforduló „gépies" feladatoknál lényegesen magasabb rendűek. A gyorsaság, a logikai döntések lehetősége, az igen nagy számú adat tárolása lehetővé teszi, hogy – természetesen a programozó által kidolgozott eljárás alapján – matematikai, műszaki, gazdasági stb. problémákban, a lehetséges kimeneteleket szimulálva mérlegeljen, és a legkedvezőbb megoldásokat megkeresse, a várható következmények elemzésével együtt.

A gondolatmenet érzékeltetésére talán alkalmas példa a személyi számítógépekkel is játszható sakkprogram, amelynek a futása során valóban az az érzésünk, mintha „gondolkozna" a gép, pedig csak az előre megírt és beolvasott program lépéseit járja végig.

Összefoglalva tehát, a számítógépek felhasználásával olyan feladatok is megoldhatók, amelyek megoldása nélkülük vagy csak nagyon rossz hatásfokkal lenne elvégezhető, vagy a megoldhatatlan feladatok közé tartozna. Gondoljunk pl. egy űrhajó adatainak meghatározására, pályájának módosítására az előző adatok alapján 10 km/s sebesség mellett, számítógép nélkül! Nézzük tehát meg egy kicsit közelebbről, hogyan is épül fel, működik egy ilyen, bonyolultságában is egyszerű eszköz, mint a számítógép.

A számítógép fő részei:

1. Központi egység (Central Processor Unit CPU)
2. Be- és kiviteli egységek (Input-Output).

Minden számítógépben a következő funkciók határozhatók meg egy adott feladat végrehajtásakor: adatbevitel, vezérlés, műveletvégzés, eredménykiíratás.

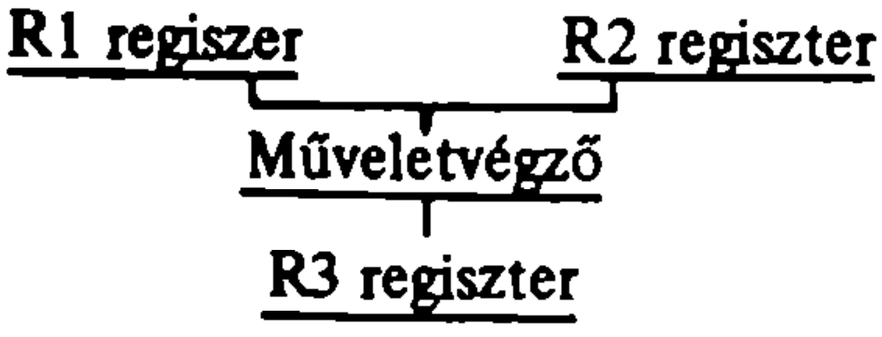
Az adatok beolvasása, az eredmények kiíratása a be- és kimeneti egységeken keresztül történik, a géptől függően kényelmesebb vagy nehézkesebb eszközökkel: lyukszalag, lyukkártya, mágnesszalag, mágneslemez stb. Így történik a végrehajtó program betöltése is.

A program feldolgozása és végrehajtása a központi egységben történik. Ennek részei, a feladataik szempontjából osztályozva:

1. aritmetikai (logikai) egység,
2. memória,
3. vezérlőegység.

AZ ARITMETIKAI EGYSÉG

Az *aritmetikai egységet* számolóegységnek vagy műveleti egységnek is nevezik a szakirodalomban. Ez a gép legkisebb egysége. Alapvető feladata az összeadás elvégzése, mivel látni fogjuk, hogy a kivonás, a szorzás és az osztás egyaránt visszavezethető az összeadásra. A műveletre váró mennyiségeket – amelyek nem okvetlenül számok, és a továbbiakban operandusoknak nevezzük őket – ideiglenes tárolókba, a regiszterekbe helyezi el a gép, és tulajdonképpen a következő egyszerű eljárást hajtja végre:



Azaz a műveletvégző elvégzi az R1 és az R2 regiszterek tartalma között az adott műveletet, és az eredményt az R3 regiszterben elhelyezni.

Az esetek többségében a műveletben résztvevő operandusok eredeti értékére nincs szükségünk, így az eredmény tárolható pl. az egyik eredeti helyén is. Ezzel kevesebb regisztert foglal egy-egy elemi művelet. A „főregisztert" akkumulátornak nevezzük, amelynek az a különleges szerepe, hogy itt történik meg az összeadás, és az eredmény is tárolódhat itt.

A gép aritmetikai egysége válaszolni tud arra is, hogy az akkumulátor tartalma egyenlő vagy nem egyenlő nullával. Ez pedig a program írásnál nagyon lényeges és jól használható, a programok elágaztatásánál van fontos szerepe.

Maga a számolómű olyan egységekből áll, amelyek a beérkező jeleket – az operandusok kódjait – átmenetileg elraktározzák, a műveletet elvégzik, és az eredményt ismét tárolják. Az első generációs gépeknél mindezt jelfogókból fölépített billenőkörökkel oldották meg. Ezért nevezzük ezeket az eszközöket elektromechanikus gépeknek. Később az ilyen áramköröket tranzisztorok felhasználásával alakították ki úgy, hogy bizonyos összetartozó egységeket (pl. összeadó áramkörök) előre összeszerelt állapotban helyeztek be, így szükség esetén a javítás, csere egyszerű volt. Természetesen a műveleti sebesség és a megbízhatóság megnőtt, hiszen az áramkörök működése nem járt együtt mechanikai mozgással. Az integrált áramkörök alkalmazásával több olyan funkcionális áramkört zsúfoltak össze működés szempontjából olyan parányi egységekbe, amelyeken belül az elemek már nem választhatók szét sem elhelyezés, sem működés szerint. 1971-től kezdődően, a mikroprocesszorok széleskörű elterjedésével és alkalmazásával, lehetőség nyílt a további miniatűrizálásra, az elemi feladatok további összevonására, és arra, hogy bizonyos belső mikroprogram beültetésével ezek az egységek szinte tetszőleges feladatokhoz igazodhassanak.

A MEMÓRIA

A központi memória (Operatív tár)

A nagy adathalmazok kezeléséhez, raktározásához az átmeneti tároló regisztereken kívül természetesen szükség van nagy kapacitású tárra is. Ez az operatív tár. Szemléletesen úgy képzelhetjük, mint egy négyzethálós papírt, amelynek minden egyes négyzetébe csak nullát vagy egyet írhatunk, és így minden sorba egy kettes számrendszerbeli számot helyezünk el. Erre a számábrázolásra később még részletesen visszatérünk.

Ahhoz, hogy a memóriában elhelyezett adatok között tájékozódni lehessen, a tárat kisebb egységekre, ún. rekeszekre osztják. Ezeket a rekeszeket megszámozzák, így kapjuk meg a címüket. Egy-egy rekesz azzal jellemezhető, hogy hány darab kétállapotú jel – bit (a bit szócska a „binary digit" – kettes számrendszerbeli számjegy – kifejezésből származik), amelynek értéke nulla vagy egy – befogadására alkalmas. A rekeszek hossza nem egységes a különböző gépeknél. Eleinte a 6-bites, majd egyre inkább a 8-bites, azaz 1-bájtos (az angol byte-ból) rekeszhossz, ill. ennek a többszöröse terjedt el. Ezeket a gépeket nevezzük „szószervezésű"-nek, ahol a „szó" hossza lehet: 16, 18, 24, 32, 60, 64 bit is.

A nagyobb szóhossz a pontosabb számábrázolást teszi lehetővé. A központi tár kapacitását a memóriában levő rekeszek száma adja meg. Ezeket 0-tól kezdve sorszámozzák, és ezek lesznek a rekeszek címei. $2^{10} = 1024$ rekeszt 1 k-val jelölünk, ahol: k olvasd kilo, és az 1024-nek a 10^3-ra való kerekítését jelenti. A központi tár általában modulszerűen építhető, bővíthető és fejleszthető. Létezik egy minimális tárméret, amely a működéshez alapvetően szükséges. Általában 8k nagyságú egységekkel bővíthető a memória, de a maximális méretnek is határt szab a gazdaságosság, a feladat-orientáltság stb. A „bájt" szervezésű gépek között pl. kisgépnek számít az, amelynek a tárkapacitása 4–32 kbájt között van, azaz ennyiszer 1024×8 bit fér el benne. A személyi számítógépek között szinte minden igényt kielégít a 64 kbájt memóriájú gép. Ma már nem ritka a több ezer kbájt feletti óriásgép sem. De a személyi számítógépek családjában is készült már 3 millió bájt kapacitású gép (1984. szeptember, IBM AT).

A központi memóriával szemben kettős követelményt kell támasztanunk. Egyrészt nagy legyen a tárolási kapacitás, másrészt rövid a *hozzáférési idő*, azaz időtartam, *amely alatt egy tetszőleges információ kiolvasható, ill. bevihető.*

Évekkel ezelőtt a legelterjedtebb, és még ma is használatos változat a ferritgyűrűs memória volt. Ebben néhány mm átmérőjű mágnesezhető fémgyűrűk kétféle mágnesezésű állapota felel meg a két lehetséges (1 és 0) bitértéknek. A gyűrűk négyzetrács pontokban helyezkednek el. Ennél a tártípusnál a hozzáférési idő kb. 0,5 és 1 μs között van. Nagy előnye, hogy az adatmozgatás nem jár együtt mechanikai mozgással, az információ be- és kivitelét a gyűrűkön átvezetett író- és olvasóvezetékek segítségével végezzük. Egyik fő hátránya – amely korlátot szab az alkalmazásának is – a nagy méret.

Az adattárolás optikai úton is történhet. Így lehetőség van arra, hogy egy 10×10 cm^2-es tárolólemezen akár százmillió információ is helyet kapjon. Ennek a módszernek a továbbfejlesztésével alakították ki a holografikus adatábrázolást, amely elsősorban a hozzáférési időt csökkentette tovább.

Még modernebb eljárással készül az ún. MOS tároló (Metal Oxid Semiconductor, azaz fémoxid félvezető típus), amelynél egy-egy parányi egység-chip több ezer információ befogadására alkalmas. Ez a fejlődés a gyorsaság, az árak csökkenése mellett a miniatürizálás irányába mutat, de bonyolultságában és sokféleségében (RAM, ROM, EPROM stb.) az amatőr érdeklődők számára szinte elérhetetlen területet jelent. Kizárólag nagy fejlesztő-

rendszerekben építhetők be, ott viszont igen változatos feladatokra, és olcsó előállítási költségek mellett.

Az integrált áramkörök további integrálódásával, a mikroszámítógépek kifejlesztése és elterjedése óriási lendületet kapott. Az LSI technológia (Large Scale Integration, azaz nagyfokú integráltság) elterjedésével újabb típusú és még előnyösebb tulajdonságú tárakat lehetett létrehozni.

A mikroelektronikában sokáig, egészen a 70-es évekig hiányoztak azok a tároló-egységek, amelyek a gyors és nem „felejtő" memóriák, valamint a lassúbb periférikus memóriák közötti tartományba eső, 10^{-3}–10^{-4} s közötti elérési időt biztosítanak. Egy lehetséges megoldást 1967-ben, az amerikai „Bell Telephone Laboratory"-ban találtak meg, a mágneses buborékmemóriában. Ez az eljárás időközben széles körben elterjedt, többek között hazánkban, a KFKI-ben fejlesztettek ki ilyen eszközöket. Működésük azon az elven alapul, hogy mágneses tulajdonságú kristályrétegben mágneses tartományokat – ún. „doméneket" – lehet létrehozni, megszűntetni és mozgatni, mikroelektronikai technológiával.

Azt hiszen mindannyian érezzük, hogy ezzel még korántsem merült ki minden lehetőség.

Például az egyik legújabb (80-as évek) félvezető technikával kidolgozott tároló chip a NVRAM, amely ugyanolyan hozzáférési sebességgel rendelkezik, mint akármelyik RAM egység, de a tárolt információt kikapcsolás után is megtartja.

Eddig csak a számítógép központi egységében levő tárakról volt szó. Közös tulajdonságuk, hogy gyors hozzáférésűek, és szemben az átmeneti tároló regiszterekkel, a rekeszek tartalmát megőrzik. A központi tár mellett, ill. azzal együtt szükség van még nagyobb kapacitású és nem okvetlenül gyors hozzáférésű külső tárakra is. A további munka során még felhasználásra kerülő információkat ui. nem érdemes egy viszonylag lassú működésű egységgel, pl. sornyomtatóval vagy képernyős megjelenítővel „kimenteni" az operatív tárból. Ez az eljárás lassú és költséges.

Ilyenkor szokás – ha rendelkezésre áll – az információkat átvinni az ún. *háttértárba*, ahol azok hosszabb ideig tárolhatók, az elérésük is gyors és egyszerű.

A háttértár feladata tehát az, hogy a pillanatnyi működéshez nem szükséges, általában nagy mennyiségű információt tárolja. Az adatok be- és kivitelére szolgáló egységekkel együtt szokás ezeket „perifériáknak" is nevezni, utalva így elhelyezésükre is.

A VEZÉRLŐEGYSÉG

A ma használatban levő gépek nagy részénél olyan szorosan összekapcsolódik a számoló- és a vezérlőegység vagy vezérlőmű feladata, hogy nem is szokás mindig különválasztani ezeket. A CPU, a központi feldolgozó egység mindkettőt magában foglalja.

Ez az egység irányítja a számítógép munkájának a menetét, biztosítja a kapcsolatot a gép egyes részei között. Értelmezi a betöltött program utasításait, amelyek általában két részből állnak, természetesen a gép számára érthető jelekkel kifejezve:

1. műveleti rész, amely meghatározza, hogy milyen típusú műveletet kell elvégezni;

2. címrész, amely azokat a regisztereket jelöli ki, amelyek tartalmával az adott műveletet végre kell hajtani.

A vezérlőmű tehát a memória megfelelő rekeszéből az operandus címét a címregiszterbe, a végrehajtandó utasítás kódját pedig az utasításregiszterbe tárolja. Ezután meg-

történik a műveletvégrehajtás, és az eredmény megjelenik a megfelelő regiszterben. Az éppen végrehajtás alatt álló utasítás címét az utasításszámláló-regiszter tárolja, amelynek tartalma a végrehajtás után 1-gyel nő.

A következő rajz szemlélteti a számítógép által elvégzett egyetlen elemi műveletet:

„C" *Címregiszter*
↓
Tár
↓
„R" regiszter
↓
Vezérlőmű → *Műveletvégző* → *Akkumulátor*

Az eljárás tehát a következő:

A „C" címregiszter tartalma szerinti címen levő operandus a memóriából átkerül az „R" regiszterbe.

A vezérlőmű az utasításregiszter tartalmának megfelelő műveletet elvégezteti a műveletvégző egységgel az „R" regiszter és az akkumulátor aktuális tartalma között. Az így kapott eredményt az akkumulátorban (főregiszter) tárolja.

Összefoglalva, a CPU akkor képes ellátni a feladatát, ha a következő kritériumoknak eleget tesz:

a) adott címről kiolvas, ill. oda adatot ír be;

b) adatokon matematikai és logikai műveleteket végez;

c) utasításokat adott sorrendben végrehajt.

A működés legfontosabb feltétele egy olyan utasításnyelv, amely a számítógép számára közvetlenül érthető módon rögzíti a program egyes lépéseit.

AZ ADATOK, INFORMÁCIÓK MEGJELENÍTÉSE

Minden információt, amelyet meg akarunk „értetni" a géppel, és szeretnénk, ha a bevitt adatokkal műveleteket végezne, le kell fordítani – kódolni – a gép mint műszaki eszköz által értelmezhető nyelvre. A használt kódrendszer akkor éri el a célját, ha a kívánt információ úgy alakítható át valamilyen jellé, hogy abból az eredeti információ egyértelműen visszaalakítható legyen. Az emberi beszéd írásban való rögzítése is egyfajta kódolás. Gondoljunk csak pl. az egyiptomi, az arab vagy a japán írásjelek „rajz-képeire"!

A kódrendszer annál könnyebben használható, minél kevesebb jelből áll a jelkészlete és ezek a jelek minél egyszerűbbek. A számítógépekkel való információ ábrázolására a legalkalmasabb, ha mindössze két lehetséges jelet engedünk meg. Az elektronikai elemek ui. a két állapotot egyszerűen képesek tárolni és feldolgozni, mégpedig az esetek többségében egy bizonyos feszültségjel-megjelenésével, ill. a jel hiányával. Később részleteiben is megismerjük az ilyen kétállapotú jelekkel kialakított kódrendszert (EBCDIC-kód).

Az ábrázolandó információ minden egyes jelét – a karaktereket – nyolc pozíción, ill. nyolc biten jelenítjük meg. Ez az ábrázolásmód szoros összefüggésben áll a kettes számrendszerrel.

Bár eddigi tanulmányaink során már használtunk a tízes számrendszertől különböző számrendszert is, emlékeztetőül tekintsük röviden át a legfontosabb fogalmakat!

A számrendszerek

A tízes vagy decimális számrendszert nap mint nap használjuk valamennyien, de szinte csak mechanikusan. Senki nem gondol pl. a 225 leírásakor arra, hogy tulajdonképpen egy összeggel, a $2\cdot10^2+2\cdot10^1+5\cdot10^0$ kifejezéssel dolgozik!

A 10 a számrendszer alapszáma (alapszám=radix, jelölése: R), az adott helyen előforduló különböző lehetséges számjegyek, az együtthatók, amelyek csak 0 és (R–1) közötti egész értékeket vehetnek föl.

Általánosságban, az R alapszámú számrendszerben egy N természetes számot, amely n számjegyből áll, a következőképpen írhatunk fel:

$$N = a_{n-1}\cdot R^{n-1}+a_{n-2}\cdot R^{n-2}+\ldots+a_0\cdot R^0 = \sum_{i=0}^{n-1} a_i\cdot R^i.$$

Megjegyzés: az a_i együtthatók nem okvetlenül különbözőek, és lehet közöttük nulla is.

A választott számrendszer alapszáma tehát az R, ami bármely 1-nél nagyobb természetes szám lehet, azaz tetszőlegesen sok alapszám alkalmas számrendszer képzéséhez. Így pl. a százas számrendszerben 100 különböző számjegy, a tizenhatos számrendszerben 16, a kettes számrendszerben 2 számjegy szerepelhet minden egyes pozíción.

Nézzük meg, melyik az a legnagyobb szám, amely egy R alapszámú számrendszerben, n pozíción megjeleníthető! Ez lesz ui. az adott n pozíciót kifejezhető legnagyobb szám, a számolás felső határa. Decimális számrendszerben, R = 10, pl. három pozíción, n = 3, a legnagyobb szám: 999, azaz 10^3-1. Ez az egyszerű szabály – gondoljuk meg! – általánosan is érvényes! $N_{max} = R^n-1$. Az R^n értéket modulusnak (M) nevezzük. M megadja az összes egymástól különböző állapotok számát. Az előbbi példánál maradva, három decimális pozíción kifejezhetjük 0 és 10^3-1 között az egész számokat, összesen 10^3 decimális számot. Például a gépkocsik kilométerszámlálóján, öt helyen, a maximális kilométerállás –N_{max}–99999, a lehetséges kilométer állások száma –M–100 000.

A digitális számítógépek működése szempontjából számunkra a legfontosabb a kettes (bináris) számrendszer!

R = 2 esetén az egyes biteken használható együtthatók, azaz a bináris számrendszer számjegyei: 0 vagy 1. Egy tetszőleges kettes számrendszerbeli szám értéke a decimális számrendszerben a következő egyszerű „számolgatással" határozható meg:

$$10011_{(2)} = 1\cdot2^4+0\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0 = 16+2+1 = 19_{(10)}.$$

Természetesen nemcsak egész, hanem törtszámok is szerepelhetnek munkánk során. Például:

$$1011.01 = 1\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0+0\cdot2^{-1}+1\cdot2^{-2} = 8+2+1+1/4 = 11{,}25.$$

A számrendszer alapszámát szokás a legkisebb helyiértékű számjegy után zárójelben jelölni. Mi ettől a továbbiakban az egyértelmű esetekben eltekintünk.

A kettes számrendszer használatának nagy előnye abból adódik, hogy minden biten vagy csak 0, vagy csak 1 állhat. Ha 0 szerepel, akkor ez a bit nulla értéket képvisel, ha pedig 1, akkor a kettő megfelelő hatványát.

Bináris számrendszerben dolgozva, pl. 8 biten, a modulus $M = R^n = 2^8 = 256$. Az ábrázolható legnagyobb szám pedig: $N_{max} = R^n-1 = 2^8-1 = 255$. Ennek megjelenítése a nyolc biten:

11111111.

Az előbbi átalakítás alapján, amelyet a számrendszerek közötti konverziónak neveznek, az is látható, hogy a decimálisan két helyiértéken fölírt szám – példánkban a 19 – binárisan öt bitet igényel, tehát lényegesen hosszabban fejezhető ki ugyanaz az érték. Ezért a számítástechnikában két másik, a 8-as (oktális), és a 16-os (hexadecimális) számrendszer is használatos, elsősorban az eredmények kiértékelésére, megjelenítésére. Az aritmetikai és logikai műveletek elvégzése, az eredmények tárolása binárisan történik.

Nyolcas (oktális) számrendszer

Egy oktális szám decimális értékét, az általános érvényű szabályok alapján így határozhatjuk meg:

$$6574_{(8)} = 6\cdot8^3+5\cdot8^2+7\cdot8^1+4\cdot8^0 = 6\cdot512+5\cdot64+7\cdot8+4 =$$
$$= 3072+320+56+4 = 3452_{(10)}$$

Látható, hogy az oktális és a decimális szám hossza és nagyságrendje között nincs lényeges különbség.

Tizenhatos (hexadecimális) számrendszer

R = 16 miatt minden pozíción 16 különböző jelet kell megjeleníteni. Erre nem elegendő a decimális számrendszer tíz számjegye. Ezért a hiányzókat betűvel jelölik, mégpedig a következő módon:

$10_{(10)} = A_{(16)}$ $\quad 11_{(10)} = B_{(16)}$ $\quad 12_{(10)} = C_{(16)}$

$13_{(10)} = D_{(16)}$ $\quad 14_{(10)} = E_{(16)}$ $\quad 15_{(10)} = F_{(16)}$

A már megismert „helyiértékes" kifejezést használva, nézzük a következő egyszerű konverziót:

$$3\ 2\ D\ 9_{(16)} = 3\cdot16^3+2\cdot16^2+13\cdot16+9 = 12288+512+208+9 = 13017_{(10)}$$

Ha összehasonlítjuk a bináris, oktális és a hexadecimális számrendszer sajátosságait, észre kell vennünk, hogy az oktálisan egy helyiértéken kifejezhető számok, azaz az oktális számjegyek binárisan három biten írhatók föl, a hexadecimális számjegyek pedig négy biten. Általában is igaz, hogy ha egy számrendszer alapszáma egy másik számrendszer alapszámának az egész kitevőjű hatványa – $16 = 2^4$, $8 = 2^3$ –, akkor a számrendszerek közötti konverzió egyszerűen elvégezhető. Ha bináris számot írunk át oktálisba, ill. hexadecimálisba, a bináris ponttól – így nevezik a tizedesvessző bináris megfelelőjét – kiindulva három bites (triéd), ill. négy bites (tetrád) csoportokra osztjuk a számot. Ezután minden egyes csoportból egy oktális, ill. egy hexadecimális számjegyet képezünk, és ezeket egymás mellé írva kapjuk az átalakított számot.

Gyakorlásképpen nézzünk néhány egyszerű konverziót!

Bináris → oktális:

1 1 1 0 1 0 1.1011 = ?

1 6 5, 5 4

Tehát $1110101.1011_{(2)} = 165{,}54_{(8)}$

Hexadecimális → bináris:

1 1 1 0 1 0 1.1011 = ?

7 5 B

Tehát $1110101.1011_{(2)} = 75{,}B_{(16)}$

Vegyük észre, hogy a hiányos triádokat, ill. tetrádokat gondolatban nullákkal töltjük fel!

Ha nyolcas számrendszerből kettesbe alakítunk, az elv ugyanaz, de az egyes számjegyeket úgy kell hármas bitekkel helyettesíteni, hogy minden bit fel legyen töltve!

Oktális → bináris:

$$6\ \ 7\ \ 2\ ,\ 4\ \ 3_{(8)} = ?$$
$$110\ 111\ 010\ .\ 100\ 011$$

Tizenhatos számrendszerből kettesbe alakításkor az egyes számjegyeket négyes bitekkel helyettesítjük.

Hexadecimális → bináris:

$$8\ \ \ A\ \ \ 5\ \ D\ \ ,\ \ 1\ \ C_{(16)} = ?$$
$$1000\ \ 1010\ \ 0101\ 1101\ ,\ \ 0001\ 1100$$

Mindezt figyelembe véve, elegendő ha a bináris–decimális átalakításban megbízható gyakorlatot szerzünk (számítógépeink is ezt teszik!), így a decimális–oktális, decimális–hexadecimális konverzió nehézkes és számolásigényes műveleteit kikerülhetjük.

Nézzük meg tehát részletesen, milyen nehézségek adódhatnak, ha egy tetszőleges decimális szám bináris megfelelőjét keressük! Az egész és törtrészt is tartalmazó számok átírásakor a legegyszerűbb, ha külön számoljuk ki az egész és külön a törtrész bináris megfelelőjét, majd ezeket a bináris ponttal egyesítjük.

Az egész rész átalakításához a következő módszer ajánlható:

1. A számot osztjuk kettővel, leírjuk a hányadost és a maradékot. A maradék lehetséges értéke 0 vagy 1.

2. Az előző osztásnál kapott hányadossal megismételjük a kettővel való osztást, mindaddig, amíg el nem érünk a nulla hányadosig. Véges sok lépésen belül ez megtörténik, mivel minden lépésnél „legalább" a felére csökken az osztandó.

3. A maradékok az utolsótól kiindulva adják a bináris szám egész részét. Tehát az utolsó hányadoshoz (amelynek értéke mindig 0) tartozó utolsó maradék (amelynek értéke mindig 1) adja a bináris szám legnagyobb helyiértékű bitjét.

Írjuk át a 121,79-et binárisba. Az egész résszel kezdve:

121 : 2

a hányadosok	*a maradékok*
60	1
30	0
15	0
7	1
3	1
1	1
0	1

A maradékok kiolvasásának iránya alulról fölfelé! Tehát a szám:

$$121_{(10)} = 1111001$$

Ha az egész részt a számítógépek esetében szokásos módon nyolc biten (8 bit = 1 bájt) írjuk föl:

$121_{(10)} = 01111001$

A törtrész átalakítása a következő szabály szerint történhet:

1. Szorozzuk kettővel a szám törtrészét, és leírjuk külön a szorzat egész részét is és törtrészét is.

2. Az előző szorzásnál kapott törtrészt ismét szorozzuk kettővel, egészen addig, amíg a szorzandó szám törtrésze nulla nem lesz. Természetesen az esetek többségében ezt a nullát nem érjük el, hiszen nem biztos, hogy 1/2 hatványaival pontosan kifejezhető a decimális szám törtrésze, de tetszőleges pontossággal megközelíthető.

3. A szorzásnál kapott egész részek (amelyek értéke 0 vagy 1) adják a decimális szám törtrészének bináris alakját, a legmagasabb helyiértékű bittől kezdődően.

	0	,	79	. 2
egész részek	1	*törtrészek*		58
	1			16
	0			32
	0			64
	1			28
	0			56
	1			122
	0			24

Az egész részek kiolvasási iránya fölülről lefelé, tehát a szám $0,79_{(10)} = 0,11001010 - 8$ bit pontossággal.

A teljes átalakítás tehát, 8–8 biten ábrázolva az egész és a törtrészt: $121,79 = 111001.11001010$. Ugyanezen szám átírása oktális, ill. hexadecimális számrendszerbe:

$121,79 = 1\ 111\ 001.110\ 010\ 10 = 1\ 7\ 1\ ,6\ 2\ 4_{(8)}$

$121,79 = 111\ 1001.1100\ 1010 = 7\ 9\ ,C\ A_{(16)}$

Gyakorlásképpen végezzük el a következő átalakításokat:

Hexadecimális → Decimális: $5A3,05_{16} = ?$ (1443,0507812)
Oktális → Decimális: $3025,17_8 = ?$ (1557,2657812)
Decimális → Bináris: $53,625 = ?$ (110101.101)

Aritmetikai műveletek bináris számrendszerben

Számítógépünk alapvető feladatai közé tartozik a bevitt és a tárolt adatokkal – operandusokkal – való művelet végzése. Ezen a területen mutatkozik meg igazán a bináris számrendszer előnye. A négy alapművelet közül ui. elegendő csak az összeadásra és a kivonásra megtanítani a gépet, mivel ebben a számrendszerben – mint ahogy ezt később látni fogjuk – a szorzás adott számú összeadásra, az osztás pedig ismételt kivonásra vezethető vissza. Sőt, a kivonás művelete is helyettesíthető összeadással, tovább egyszerűsítve ezzel a számítógép műszaki kialakítását.

Mielőtt rátérnénk a bináris műveletek tanulmányozására, alapvetően fontos, hogy

megismerkedjünk a Boole-algebrával, a kétállapotú mennyiségek algebrájával. Alapjait George Boole dolgozta ki a múlt században, de a számítógépek működésében ma is nélkülözhetetlen. Általános tételeket és szabályokat tartalmaz, függetlenül attól, hogy a kétféle jel – állapot – milyen jelentéssel rendelkezik. Ha a két lehetséges érték az 1 és a 0, a kettes számrendszerre ad jól használható és egyértelmű információkat.

A két értéknek tulajdoníthatunk olyan jelentést, amely egy esemény bekövetkezését vagy be nem következését jelenti. Ilyenkor azt mondjuk, hogy az adott eseményhez rendelünk egy logikai változót – logikai, mert ezekkel az eseményekkel aritmetikai műveletek nem végezhetők, de amint ezt hamarosan látjuk, logikai műveletek igen –, amelynek logikai értéke bekövetkezéskor „igaz", ellenkező esetben „hamis". Az igaz, ill. a hamis értéknek megfeleltethetjük az „1"-et, ill. a „0"-át.

Pl. a) Megirom egy feladat számítógépes programját, tehát az eseményhez rendelt logikai változó értéke 1, ha;

b) Van az iskolában számítógép – ha van 1, ha nincs 0, és ha

c) Tudok programozni. Ennek a változónak a lehetséges értékei 1 vagy 0.

Tehát az a) változó értéke 1, ha a b) és a c) változók értéke is 1.

Így ha nem is jelentésben, de jelölésben, műveletekben szoros analógia figyelhető meg a „bináris" algebra és a „logikai" algebra között. Lényeges különbség az, hogy a bináris algebrában a változóknak numerikus jelentésük van, és lehetséges értéküket egy csupa 0-ból és 1-ből álló jelsorozat képviseli, a Boole-algebrában pedig a változóknak csak logikai jelentése van, és két lehetséges értéke, amelyeket csak jelölünk 0-val vagy 1-gyel.

A logikai algebra tulajdonképpen az események, azaz a logikai változók között végezhető műveletekkel foglalkozik. Az eredmény szintén csak kétértékű lehet.

Logikai műveletek:

Ezek végeredményét, azaz az adott művelet „bemenő" és „kimenő" adatai között az összefüggést ún. igazságtáblázatokkal szemléltetjük.

„NEM" művelet, logikai tagadás. Egy „A" esemény tagadását, amelyet „$\bar{A}$"-val jelölünk, a következő egyszerű igazságtáblázat tartalmazza:

$$A = 1, \rightarrow \bar{A} = 0$$
$$A = 0, \rightarrow \bar{A} = 1$$

Tehát az eredeti logikai értéket minden esetben az ellentettjére kell változtatni. Ezt nevezik invertálásnak.

„ÉS" művelet, logikai szorzás. Itt két vagy több bemenő esemény és egy kimenő szerepel. A műveletet a szorzás jelével jelöljük: $X = A \cdot B$. Az egymással ÉS kapcsolatban levő bemenő változók mindegyikének „1" értéket kell felvenni, csak akkor lesz „1" az eredmény. Így az igazságtáblázat:

$$A = 1\ ,\ B = 1\ ,\ \rightarrow\ X = 1$$
$$A = 1\ ,\ B = 0\ ,\ \rightarrow\ X = 0$$
$$A = 0\ ,\ B = 1\ ,\ \rightarrow\ X = 0$$
$$A = 0\ ,\ B = 0\ ,\ \rightarrow\ X = 0$$

„VAGY" művelet, logikai összeadás. Szintén kétváltozós művelet, amelynek jelölése: $X = A + B$. Azt tartalmazza, hogy valamelyik esemény bekövetkezése már maga után vonja az eredmény igaz értékét. Az igazságtáblázata:

$A = 1\ ,\ B = 1\ ,\ \rightarrow\ X = 1$
$A = 1\ ,\ B = 0\ ,\ \rightarrow\ X = 1$
$A = 0\ ,\ B = 1\ ,\ \rightarrow\ X = 1$
$A = 0\ ,\ B = 0\ ,\ \rightarrow\ X = 0$

Nem tartozik szorosan a logikai alapműveletek közé, de gyakran használatos a *„kizáró VAGY"* művelet, amely annyiban tér el a VAGY-tól, hogy a bemenő események egyidejű bekövetkezésekor 0 értéket vesz föl. Tehát ha A = 1 és B = 1 egyszerre teljesül, akkor X = 0. Jelölése: +

A három logikai alapművelettel tetszőleges további logikai összefüggés fölépíthető. Például:

$$A+B = \bar{A}\cdot B+A\cdot\bar{B}$$

Ez az állítás könnyen belátható, akár az igazságtáblázatok összevetésével, akár a matematikában ismert halmazelméleti megfontolással, ahol a logikai változóknak egy-egy síkbeli ponthalmazt feleltetünk meg.

Az összetett logikai kifejezések átalakítása az ún. de Morgan azonosságok alapján végezhetők el:

$$\bar{A}+\bar{B} = \overline{A\cdot B} \qquad \bar{A}\cdot\bar{B} = \overline{A+B}$$

Ezek igazolását az olvasóra bízva, megjegyezzük, hogy a logikai áramkörök tervezésénél jól használhatók ezek az „egyszerűsítő" átalakítások.

Logikai műveleteket nemcsak logikai változókkal végezhetünk, hanem tetszőleges kétállapotú adatokkal is. Ilyenkor az operandus egyes bitjeivel egymástól függetlenül kell elvégezni az adott műveletet.

Végezzünk pl. logikai VAGY-ot a következő két hexadecimális szám, ill. bináris megfelelőik között:

„A" = B 2 F, „B" = 685, „A"+„B" = ?

$$\begin{array}{r} 1011\ 0010\ 1111 \\ +\ 0110\ 1000\ 0101 \\ \hline 1111\ 1010\ 1111 \end{array}$$

Tehát az eredmény: FAF. Ahogy azt majd hamarosan látni fogjuk, ez az eredmény majdnem megegyzik a bináris összeadással kapott eredménnyel.

ÉS műveletet végezve az előbbi két operandussal, szintén bitenként,

„B2F"·„685" = ?

$$1011\ 0010\ 1111 \cdot 0110\ 1000\ 0101 = 0010\ 0000\ 0101$$

Az eredmény hexadecimálisan fölírva: 2 0 5.

Az említett logikai azonosságok és az alapműveletek tulajdonságainak felhasználásával összetett logikai kifejezések egyszerűbb alakra hozhatók. Például:

$$\bar{A}\cdot(A+B) = \bar{A}\cdot B$$
$$A+(A\cdot B)+(\bar{A}\cdot B) = A+B$$
$$A+\bar{A} = 1$$
$$A\cdot\bar{A} = 0$$

Meg kell végezetül említeni még a relációkat, mivel ezekkel gyakran találkozunk a programozás során. Ilyenkor két, nem okvetlenül algebrai kifejezés viszonyát vizsgáljuk, amelyek lehetnek: kisebb, nagyobb vagy egyenlő.

Nézzük tehát hogyan végez számítógépünk aritmetikai műveleteket!

Bináris összeadás

Mivel a gép összeadó egysége is, még több összeadandó esetében is egyszerre csak két operandus között végzi el a műveletet, mi is ezt az egyszerű esetet tanulmányozzuk. Minden helyiértéken – biten – a következő lehetséges esetek fordulhatnak elő:

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ +0 & +0 & +1 & +1 \\ \hline 0 & 1 & 1 & \overset{1}{-}1 \end{array}$$

A megjelölt 1-es „átvitelt" jelent a következő, 1-gyel magasabb helyiértékű bitre, és ezt ott természetesen összeadandónak kell tekinteni. Az átvitellel keletkezett 1-es megfelel a decimális számrendszerben a tízes átlépésnek. Több számjegyű számok összeadásánál tehát ügyelni kell az egyes biteken kialakuló részeredményre és az alacsonyabb helyiértékről esetleg átkerülő átvitelre is.

Nézzünk erre is példát úgy, hogy eredményeinket „ellenőrizzük" decimálisan!

$$\begin{array}{rr} 101110 & 46 \\ +\ 10111 & +23 \\ \hline 1000101 & 69 \end{array}$$

Az előbbi példából látható, hogy előfordulhat, hogy az összeg nem fér el annyi biten, mint az összeadandók. Ebben az esetben keletkezhet az ún. „túlcsordulás", amelyre később még visszatérünk.

Bináris kivonás

A tízes számrendszerben megszokott és tetszőleges számrendszerben alkalmazható módszer kényelmesen használható ugyan kettes számrendszerben is, azonban a számítógépek belső kiépítése szükségessé tette egy másik eljárás kialakítását. Az alapvető elv az volt, hogy az alapműveleteket egymásra visszavezethetően lehessen elvégezni! Ezért használják a kivonás elvégzésére az ún. „komplemens aritmetikát", amely lehetővé teszi, hogy a kivonást is az összeadó egység végezze el. A komplemens szó két egymást kölcsönösen kiegészítő dolog egyikét jelenti. Mit kell kivonáskor kiegészíteni, és mire?!

A kivonandót, azaz a negatív számot, az adott pozíciószámhoz tartozó modulusra, és így már végezhető is az összeadás.

Nézzük ezt meg részletesen! Például 8 biten (1 bájt) a -5 megjelenítése a következő módon történik komplemens képzéssel: $-5 = ?$

1. $5 = 101 = 0\,0\,0\,0\,0\,1\,0\,1$

2. Invertálás, azaz az 1 értékű biteket 0-ra, a 0 értékűeket 1-re cseréljük: $00000101 \rightarrow \quad 1\,1\,1\,1\,1\,0\,1\,0$

3. Az így nyert számhoz 1-et hozzáadunk, mivel a modulus 1-gyel nagyobb mint a legnagyobb ábrázolható szám. Tehát: $-5 = 1\,1\,1\,1\,1\,0\,1\,1$

Ezek alapján a számítógép 8 biten a $22-5 = 22+(-5) = 17$ egyszerű műveletet így végzi el:

$$\begin{array}{rl} 22: & 0\,0\,0\,1\,0\,1\,1\,0 \\ -\ 5: & \underline{1\,1\,1\,1\,1\,0\,1\,1} \\ 17: & \overset{1}{-}0\,0\,0\,1\,0\,0\,0\,1 \end{array}$$

A 8 biten keletkező eredmény a helyes értéket adja meg. Az első pozíción, azaz a kilencedik biten keletkező 1-es értéket nem kell figyelembe venni. Ezt a gép is „elfelejti", hiszen csak nyolc biten számol! Ez természetesen nem azonos a túlcsorduláskor keletkező 1-gyel, hiszen a nyolcadik pozíción álló 0 azt is jelenti, hogy az eredmény pozitív előjelű, nem kell „visszaalakítani".

Vizsgáljuk meg azt az esetet is, amikor az eredmény is negatív lesz.

$75-96 = 75+(-96) = -21$

75 :	01001011
96 :	01100000
invertálás	10011111
+1: → −96:	10100000
eredmény:	11101011

A legmagasabb helyiértékű biten – ez az előjel bit – álló 1-es a negatív előjelre utal, így az eredményt még vissza kell alakítani, rekomplementálni:

$$11101011 \rightarrow 00010100+1 \rightarrow 00010101 = 21$$

Tehát a helyes eredmény az előjellel együtt: −21.

Lehetséges, hogy ez az eljárás első olvasásra nehézkesnek tűnik, de ha meggondoljuk, felhasználásával a számítógépben csak összeadó és invertáló egység szükséges a négy alapművelet – hamarosan látjuk, hogy a szorzás és az osztás is elvégezhető ilyen egységgel – elvégzéséhez. Belátható, hogy ez már megéri a fáradságot.

Természetesen a komplemens aritmetika használható decimális számrendszerben is előjeles számok összevonására, de ott az invertálás a számjegyek 9-re történő kiegészítését jelenti. Kipróbálni lehet, sőt érdemes, de használni jobb a „hagyományos" kivonást!

Mivel a számítógépek nagy része az adatokat 32 biten, azaz 4 bájton tárolja (ezek a „szószervezésű" gépek), nézzünk még egy feladatot előjeles számok összeadására, 32 biten, hexadecimális alakban, amelynél 32 bit = 1 szó áll rendelkezésünkre egy-egy szám ábrázolásához. (Általában lehetőség van félszó és duplaszó használatára is.) $179-108 = 71$.

179: 00...0 10110011 = 000000 B 3
(24 nulla)

108: 00...0 01101100

−10: 11...1 10010100 = F F F F F F 9 4
(24 egyes)

179−108: 00...0 10110011
11...1 10010100
100...0 01000111 = 00000047

A kapott szám pontosan +71-nek a hexadecimális alakja:

$$00000047_{16} = 71_{10}$$

Amennyiben az eljárás érthető, ellenőrizzük tudásunkat a következő két feladat megoldásán!

1. Egy művelet során, 32 biten, előjelesen a következő eredményt kaptuk: $FFFFF3C8_{16}$. Értelmezzük az eredményt, milyen decimális számnak felel meg?

Megoldás:
F F F F F 3 C 8 = 1 1 ... 1 0 0 1 1 1 1 0 0 1 0 0 0
Az „előjelbit" 1-es, tehát negatív számról van szó,
invertálni kell: 0 0 ... 0 1 1 0 0 0 0 1 1 0 0 0
Ez a szám hexadecimális számrendszerben:
$C\ 3\ 8_{16} = 3128_{10}$
Tehát a végeredmény az előjelet is figyelembe véve:
F F F F F 3 C 8 = –3128

2. 32 biten melyik az előjelesen ábrázolható, legnagyobb abszolút értékű negatív szám?

Megoldás:
32 biten a legnagyobb pozitív szám, az előjelbit figyelembevételével:

0 1 . . . 1
(29 egyes)
Ezt invertálva megkapjuk a kérdéses számot:

$1\ 0\ 0\ldots0\ 1 = 8\ 0\ 0\ 0\ 0\ 0\ 0\ 1_{(16)}$
(28 nulla)

Szorzás és osztás elvégzése kettes számrendszerben

Amint már jeleztük, ez a két művelet is visszavezethető összevonásra. Az eljárás teljesen megegyezik a decimális számrendszerben használttal, ezért csak egy-egy példán keresztül vizsgáljuk meg!

Szorzás: pl. 11·13 = 143
binárisan: 1 0 1 1 · 1 1 0 1
1 0 1 1
1 0 1 1
1 0 0 0 1 1 1 1 = 143

Osztás: pl. 117:22 = 5,3181
Előbb számítsuk ki az egész részt:

binárisan: 1 1 1 0 1 0 1 : 1 0 1 1 0 = 1 0 1 ...
1 0 1 1 0
0 1 1 1 0 1
1 0 1 1 0

Elvégezve az utolsó kivonást, folytathatjuk a törtrész kiszámításával:

1 1 1 0 0 : 1 0 1 1 0 = .0 1 0 1 0 0 0 1
1 0 1 1 0
1 1 0 0 0
1 0 1 1 0
1 0 0 0 0 0
1 0 1 1 0
1 0 1 0

Ez az utolsó maradék, a számolást tovább folytathatjuk, tetszőleges pontosságig. Tehát az eredmény: 101.01010001 = 5,318.

Feltehetően a két előbbi példából látható, hogy a szorzás valóban visszavezethető egymáshoz képest sorozatosan eltolt összeadásokra, az osztás pedig annak az eldöntése, hogy egy adott szám a másikból hányszor vonható ki. Így a négy alapművelet elvégzésére alkalmas egységnek, az aritmetikai egységnek nem kell mást „tudnia ", mint az előjel bitet is beleértve, bitenként összeadni, jobbra, ill. balra eltolni (ez a siftelés) és invertálni. A gépi szintű programozásnál még visszatérünk erre a kérdésre. Hogy az ilyen egységeket hogyan valósítják meg ma a gyakorlatban, annak tárgyalására ebben a könyvben nincs mód.

ADATHORDOZÓK

A számrendszerekben végzett barangolásunk után kanyarodjunk most vissza kiindulási kérdésünkhöz, a számítógépben megjelenítendő adatok ábrázolásához.

A számítógépek többsége, különösen a személyi számítógépek olyan műszaki adottsággal – hardver (angolul: hardware) – rendelkeznek, amely minden információt kétállapotú jelek sorozatával tárol. Ezért is foglalkoztunk, bár a teljessség igénye nélkül, de részletesen a bináris számrendszerrel és a Boole-algebrával. A személyi számítógépek közül sokban a nemzetközi szabvány szerint a nullának a +12V, az egynek a–12V felel meg. A belső karakter ábrázolás legelterjedtebb módja az EBCDIC-kód. Ez a rövidítés egy angol kifejezés kezdőbetűit tartalmazza: *E*xtended *B*inary *C*oded *D*ecimal *I*nterchange *C*ode. Magyarul: A binárisan kódolt decimális számok kiterjesztett átalakító kódja. 8 biten (1 bájt) kerül minden karakter ábrázolásra, azaz a modulus: M = 256. Tehát ez a kód alkalmas 256 féle jel kifejezésére! Ez olyan nagy szám, hogy ennyi karakter fölösleges a számítógépek egyszerű nyelvén. Ezért lehetőség van bizonyos egyszerűsítő, a megjegyzést megkönnyítő összevonásokra is. A 8 bit fel van osztva két 4-bites tetrádra, két félbájtra:

– – – – / – – – –

zóna bitek numerikus bitek

Az így létrejövő 1–1 félbájt tartalma éppen 1–1 hexadecimális számjeggyel reprezentálható. Ennek alapján a nagybetűk (alfabetikus jelek) EBCDIC kódjai hexadecimális alakban a következők:

A : C1, B : C2, ... I : C9,
J : D1, K : D2, ... R :D9,
S : E2, T : E3, ... Z :E9.

Az „egyéb" karakterek kódjai, amelyek általában az adott gép leírásában megtalálhatók, szintén „belefférnek" a 256 lehetőségbe. Például a szóköz kódja: 40, a (+): 4E, a (–): 60.

A decimális számjegyek kódjai:

0 :F0, 1 :F1 . . . 9 :F9.

Ilyen alakban azonban a zónabitek miatt a gép nem tud aritmetikai műveleteket végezni ezekkel a számokkal. Csak a „szövegszerű" információk kezelésénél használható

ez a kódolás, ahol a „szöveg", amely tartalmazhat betűket, számokat és egyéb karaktereket, annyi bájtot foglal el, ahány karakterből áll.

Például a „VW GOLF" kifejezés EBCDIC formája a következő: E5 E6 40 C7 D6 D3 C6.

Az aritmetikai műveletekhez is használható numerikus adatok ábrázolásának egyik közismert módja a BCD kód. A BCD (*B*inary *C*oded *D*ecimal) jelentése: binárisan kódolt decimális szám. Ez az ábrázolási forma azt tartalmazza, hogy decimális számrendszerben marad az adott szám, de minden egyes számjegyét bináris számrendszerben írjuk fel. A tíz decimális számjegy kódolásához négy bit szükséges, így 1 bájton két BCD számjegy fér el. Ez a forma az ún. tömörített alak. A szám előjele a legkisebb helyiértékű félbájton kerül kifejezésre, és mivel félbájt nem maradhat üresen, a páros számú számjegyből álló számok esetében a legnagyobb helyiértékű félbájtot 0000-vel kell feltölteni.

Az előjel kódja: (+):1100 = C
(–):1101 = D

Így pl. 1984 kódja:

0000 0001 1001 1000 0100 1100 = 0 1984C.

A tömörített alakban kódolt operandusokkal bizonyos típusú számítógépek képesek aritmetikai műveleteket végezni, de csak azok, amelyek rendelkeznek „decimális aritmetikai egységgel". Ebben az esetben a számok előjelét, azaz az utolsó félbájtot külön kell választani, és az eredmény előjelét megállapítani. A másik nehézség, hogy ebben a formában nincs lehetőség a tizedestörtek használatára, tehát csak egész típusú adatoknál használható. Éppen ezért a BCD kód elsősorban olyan gépeknél használatos, amelyek alkalmazási területe az adatfeldolgozás, az ügyvitelszervezés, ahol többnyire elegendő a négy alapművelet elvégzése egész típusú változókkal.

A művelet befejezése után az eredmény tárolása, ill. kiíratása is úgy lenne könnyen kezelhető, ha egy-egy bájt tartalma egy-egy karakter EBCDIC kódjaként jelentkezne. Ehhez azonban a tömörített alakot szét kell húzni, hiszen ott egy bájton két számjegy jelenik meg. Ez az ábrázolás a zónás alak. Minden decimális számjegyet egy teljes bájton ábrázolunk, kivéve a legalacsonyabb helyiértéken állót, mivel ott az egyik félbájton továbbra is az előjel kap helyet. A számjegyeket tartalmazó bájtok első fele, a zónarész 1111-gyel van kitöltve. Így a zónás alakban kódolt decimális szám alakja a következő:

(zónarész) (BCD számjegy)(zónarész)(BCD számjegy) . . . (előjel) (számjegy)

Az előjel kódja itt is a már megismert: (+) :1100 = C
(–):1101 = D.

Példaként nézzük meg 1080 BCD kódját, zónás alakban:

11110001 11110000 11111000 11000000 = F 1 F 0 F 8 C 0.

Látható, hogy ez az alak már csak az utolsó bájt tartalmában különbözik az adott szám EBCDIC alakjától. A mai gépeknél automatikusan végbemegy az átalakítás, amelynél az előjel kikerül a szám elé, az utolsó előtti félbájt pedig 1 1 1 1 = F-fel töltődik fel.

Eddigi ismereteink alapján valmennyi, a programozás során felhasználásra kerülő karakter, numerikus, alfabetikus és egyéb karakterek, felírhatók és felismerhetők egységes

formában, EBCDIC kódban. A betű- és egyéb karakterek ábrázolására elterjedt és használatos ez a forma. A numerikus adatokhoz azonban sokkal inkább a bináris számrendszerbeli ábrázolást használják. Ekkor nem csupán a decimális számjegyek átírását jelenti a kódolás, hanem a teljes szám konvertálását bináris rendszerbe. Valamennyi ma használatban levő gép ilyen alakban ábrázolja és dolgozza fel az adatokat, mégpedig kétféle módon: fixpontos és lebegőpontos formában.

Fixpontos számábrázolás

Ez a számábrázolás csak egész számok esetén használható. A gép általában az utolsó számjegy után helyezi el a képzeletbeli bináris pontot. A rendelkezésre álló terület minden gépnél adott.

Maradjunk jelenleg a 32 pozíción történő megjelenítésnél. A legnagyobb helyiértékű bit az előjelbit. A már korábban tárgyalt komplemens aritmetika használata miatt a pozitív előjelet a nulla, a negatívat az 1 jelenti. Kötelező szabály a „jobbra tömörítés", tehát az előjelbit és a bináris szám közötti pozíciókon csupa nulla áll.

Például a 39-es szám ábrázolása fixpontosan, 32 biten:

$$0\,0\ \ldots\ 0\,1\,0\,0\,1\,1\,1 = 0\,0\,0\,0\,0\,0\,2\,7_{16}$$

Az egy „szó"-ban kifejezhető legnagyobb pozitív szám:

$$N_{max.} = 2^{31}-1 = 2147\,483\,647 = 7\,F\,F\,F\,F\,F\,F\,F_{16}$$

A negatív számok ábrázolása a korábban megismert komplemens aritmetika alapján történik, azaz invertálunk, majd 1-et hozzáadunk. Például $-100 = ?$

1. $+100 = 1\,1\,0\,0\,1\,0\,0$
2. $-100 = 0\,0\,1\,1\,0\,1\,1+1 = 0\,0\,1\,1\,1\,0\,0$

Mindez 32 biten; az invertálást minden biten végrehajtva:

$$1\,1\ \ldots\ 1\,0\,0\,1\,1\,1\,0\ \ 0 = F\,F\,F\,F\,F\,F\,F\,C$$

(25 egyes!)

A fixpontos számábrázolás nagy előnye, hogy az alapműveletek így már mind visszavezethetők az előjellel együtt végrehajtandó összeadásra. Az előjelbitet csak az eredmény előjelének értelmezésénél kell külön kezelni. Végezzük el 32 biten a következő műveleteket!

$60-29 = ?$

60	:	$0\,0\ \ldots\ 0\,1\,1\,1\,1\,0\,0 = 0\,0\,0\,0\,0\,0\,3\,C$
29	:	$0\,0\ \ldots\ 0\,0\,1\,1\,1\,0\,1 = 0\,0\,0\,0\,0\,0\,1\,D$
-29	:	$1\,1\ \ldots\ 1\,1\,0\,0\,0\,1\,1 = F\,F\,F\,F\,F\,F\,E\,3$
60–29	:	$0\,0\ \ldots\ 0\,1\,1\,1\,1\,0\,0$
		$1\,1\ \ldots\ 1\,1\,0\,0\,0\,1\ \ 1$
		$0\,0\ \ldots\ 0\,0\,1\,1\,1\,1\,1 = 0000001F(=31_{10})$.

Abban az esetben, ha két olyan nagy azonos előjelű számot vonunk össze, hogy az eredmény nem ábrázolható az adott számú pozíción, a végeredmény hibás lesz. Ez a jelenség a túlcsordulás, amelyet az előjelbit indokolatlan megváltozása jelez. Példa erre a következő összeadás:

0 1 0 1 1 1 0 . . . 0 1 0 1 ≐ 9 C 0 0 0 0 0 5
+0 0 1 1 0 0 1 . . . 0 0 1 1 ≐ 3 3 0 0 0 0 0 3

1 0 0 0 1 1 1 . . . 1 0 0 0 (a pontozott helyen 0 áll).

Az előjelbit indokolatlanul negatív számot jelez, tehát az eredmény helytelen, hiszen két pozitív számot adtunk össze. Általában a számítógép programfutás közben „figyeli" a túlcsordulást, és jelzi.

Lebegőpontos számábrázolás

Mivel a fixpontosan ábrázolt számoknál az értékhatár meghatározott, és csak egész számok ábrázolhatók így; azok amelyeknek abszolút értéke $2^{n-1}-1$-nél kisebb, ha „n" bites eszközzel dolgozunk, nem minden esetben megfelelő. Ennél általánosabb és rugalmasabb a lebegőpontos ábrázolás. Főleg a számításos feladatoknál elkerülhetetlen a törtszámok használata. Ezek kódolásánál a matematikában jól ismert normálalakból indulunk ki. Minden A valós szám fölírható a következő alakban: $A = X \cdot R^Y$. Ezt az átírást nevezzük normalizálásnak, ahol A az eredeti szám; X az együttható vagy mantissza; R a számrendszer alapszáma; Y a kitevő vagy *karakterisztika.*

A számítástechnikában a normálalak annyiban különbözik a matematikában megszokott normálalaktól, hogy itt a mantissza nem 1 és 10 közé esik, hanem mindig 0 és 1 közé. Ennek az az előnye, hogy a mantisszában nem kell tizedes vesszőt vagy bináris pontot stb. használni, az automatikusan a szám elé kerül. A karakterisztika, amely mindig egész szám, tulajdonképpen a tizedesvessző (bináris pont stb.) eredeti helyéhez képest az eltolás lépéseinek a számát jelenti. A karakterisztika előjele pedig megadja az eltolás irányát. Tekintsünk egy-két példát normál alakra!

10-es számrendszerben:

$+1948 = +0{,}1948 \cdot 10^4$; X = 1948, Y = 4
$+0{,}001948 = +0{,}1948 \cdot 10^{-2}$; X = 1948, Y = –2

Kettes számrendszerben:

$+1010.001 = +.1010001 \cdot 2^4$; X = 1010001, Y =4
$-0.101 = -\ 101 \cdot 2^0$; X = 101, Y = 0

Tizenhatos számrendszerben:

$+9C{,}3FA = +{,}9C3FA \cdot 16^2$, X = 9C3FA, Y = 2.

Amennyiben az alapszámot előre meghatározzuk, mint ahogy a ma használatos számítógépeknél 16-osra normalizálnak, akkor az eredeti A számot egyértelműen megadhatjuk az előjelével, a mantisszával és az előjeles karakterisztikával. Az ábrázolás hossza most is megegyezik a gép által használt szó hosszával. Így pl. 32 biten a szokásos ábrázolási forma:

1 bit előjel, 7 bit előjeles karakterisztika, 24 bit mantissza.

Az előjel – ami a matissza és egyben a szám előjele is – megjelenítése a már megismert módon történik:

A pozitívat jelenti a 0, a negatívat pedig az 1. A karakterisztika, az előjeles kitevő kódolására egy különleges módszer, az alapértékes ábrázolás használatos. 7 pozíció áll rendelkezésre, ezen 128 különböző állapot fejezhető ki. Ennek az értéknek a

fele, 64, jelenti a kitevőt, ez az „alapérték". A pozitív kitevőt ehhez hozzáadjuk, a negatív kitevő értékét pedig ebből levonjuk. Mivel a hét biten megjeleníthető maximális szám: $1111111 = 127_{10}$, és ez a 1000000 alapértéknél $0111111 = 63_{10}$-mal nagyobb, így a maximális kitevő: 63. A legkisebb kitevő, azaz a legnagyobb abszolút értékű negatív szám nagyságrendje amely még ábrázolható, úgy kapható meg, ha az alapértékből a lehetséges legnagyobb számot: $1000000 = 64_{10}$-et vonunk le. Tehát az ábrázolható legnagyobb negatív kitevő: -64.

A mantisszát mindig binárisan ábrázoljuk. Ha rövid vagy egyszeres pontosságú lebegőpontos alakot használunk, akkor egy szóhosszon (32 bit) a mantissza számára 24 bit jut. Dupla pontosságú számábrázolásnál – erre a számítógépek többségénél általában van lehetőség – a karakterisztika változatlan, a mantissza viszont $64-8 = 56$ bitre kerülhet.

Ezek után nézzük meg pl. -75 rövid lebegőpontos ábrázolását, 16-osra normalizálva:

$$-75 = -1001011 = -4B = -0{,}4B \cdot 16^2.$$

1. negatív előjelbit: 1;
2. a karakterisztika +2, tehát az alapértékhez hozzáadjuk: 1000010;
3. a mantissza: 01001011 = 4B.

Tehát a teljes szám 32 biten:

$$1\ 1000010\ 01001011\underbrace{0\ldots0}_{\text{(16 nulla)}}$$

Mivel kiíratáskor a gép is így jár el, írjuk át mi is hexadecimális alakra: (balról kiindulva képezünk tetrádokat)

$$-75 = \text{D 2 4 B 0 0 0 0}.$$

Gyakorlásképpen ábrázoljuk még a 0,00125-öt ugyanígy: $0{,}00125 = 0{,}000000000101 = 0{,}005_{16} = 0.5 \cdot 16^{-2}$ ha 16-osra normalizálva írjuk fel.

A karakterisztika -2, tehát levonjuk az alapértékből:

$$1000000-1 = 64-2 = 62 = 0111110.$$

Tehát a szám rövid lebegőpontos alakja:

$$0\ 0111110\ 01010\ldots0 = 3D500000.$$

Bevezetés a programozásba

Az eddig elmondottakból feltételezhetõen mindenki számára kiderült, hogy a számítógép akkor képes a felhasználó által kitűzött feladatot „automatikusan" megoldani – azaz a bemenő adatok alapján az eredményt meghatározni –, ha ismeri a feladat hibátlan programját. Azaz, ha mi elkészítjük az adott probléma megoldásához elvezető algoritmust.

Az algoritmus tehát egy olyan eljárás, amely egy nem okvetlenül matematikai feladat véges sok elemi lépésben való optimális megoldására irányul, amennyiben van megoldás! A valamennyire is összetett feladatokat elemi lépésekre kell bontani, amely lépések során vagy megfelelő értékeket rendelünk változókhoz, vagy a változók aktuális értékétõl függõen a feladat további lépéseinek lehetséges menetét határozzuk meg. Az ilyen elágaztatási pontoknál felmerülő döntések értelmezéséhez bizonyos logikai ítéletek igazságát kell megvizsgálni, és annak értelmében továbblépni.

Ha egy algoritmus már a számítógép által értelmezhető lépésekből, utasításokból áll, akkor nevezzük programnak. (Ennek különböző szintjei: gépi nyelven, assembly szinten és magas szintű programnyelven írt.)

Ahhoz, hogy programunk „fusson" a gépen, azaz, hogy a gép helyesen felismerje az általunk közölt utasításokat, valamint, hogy azok önmagukban és a feladat egészét tekintve is helyesek legyenek, lényeges tulajdonságokkal kell rendelkeznie:

1. Pontos és részletes leírását tartalmazza azoknak a műveleteknek, amelyek a bemenő adatokból a kimenő adatokat előállítják, beleértve a műveletek végrehajtási sorrendjét is.

2. A program a számítógép által értelmezhető nyelven megfogalmazva tartalmazza ezeket az utasításokat.

Egyelőre válasszuk külön a két szempontot, és csak az elsővel foglalkozzunk. A programnyelvre a későbbi fejezetekben teljes részletességgel kitérünk.

A program megtervezéséhez, az algoritmus egyes lépéseinek a rögzítéséhez általánosan és jól használható módszer a blokkdiagram készítése. Ez egy jelölésrendszer, amely bizonyos egyszerű geometriai jelekből, és az ezekben elhelyezett matematikai és logikai leírásokból áll. A blokkdiagramban levő egyes lépések a soronkövetkező elemi lépést definiálják. Azok az elemi lépések, amelyek már a gép által végrehajthatók, döntõen kétféle típusúak lehetnek:

1. Az értékadó vagy általános művelet, amelynek egy téglalap a jele. Ebbe írjuk a végrehajtandó műveletet ill. tevékenységet, matematikai kifejezéssel vagy szövegesen.

2. A döntési művelet, amelyet rombusszal jelölünk, és ez tartalmazza az eldöntendő kérdést. Ha a kérdés logikai jellegű, akkor kétféle irányban léphet tovább a feladat, a lehetséges válasz ui. a kérdésre „igen" vagy „nem". Ha a kérdés matematikai jellegű, a

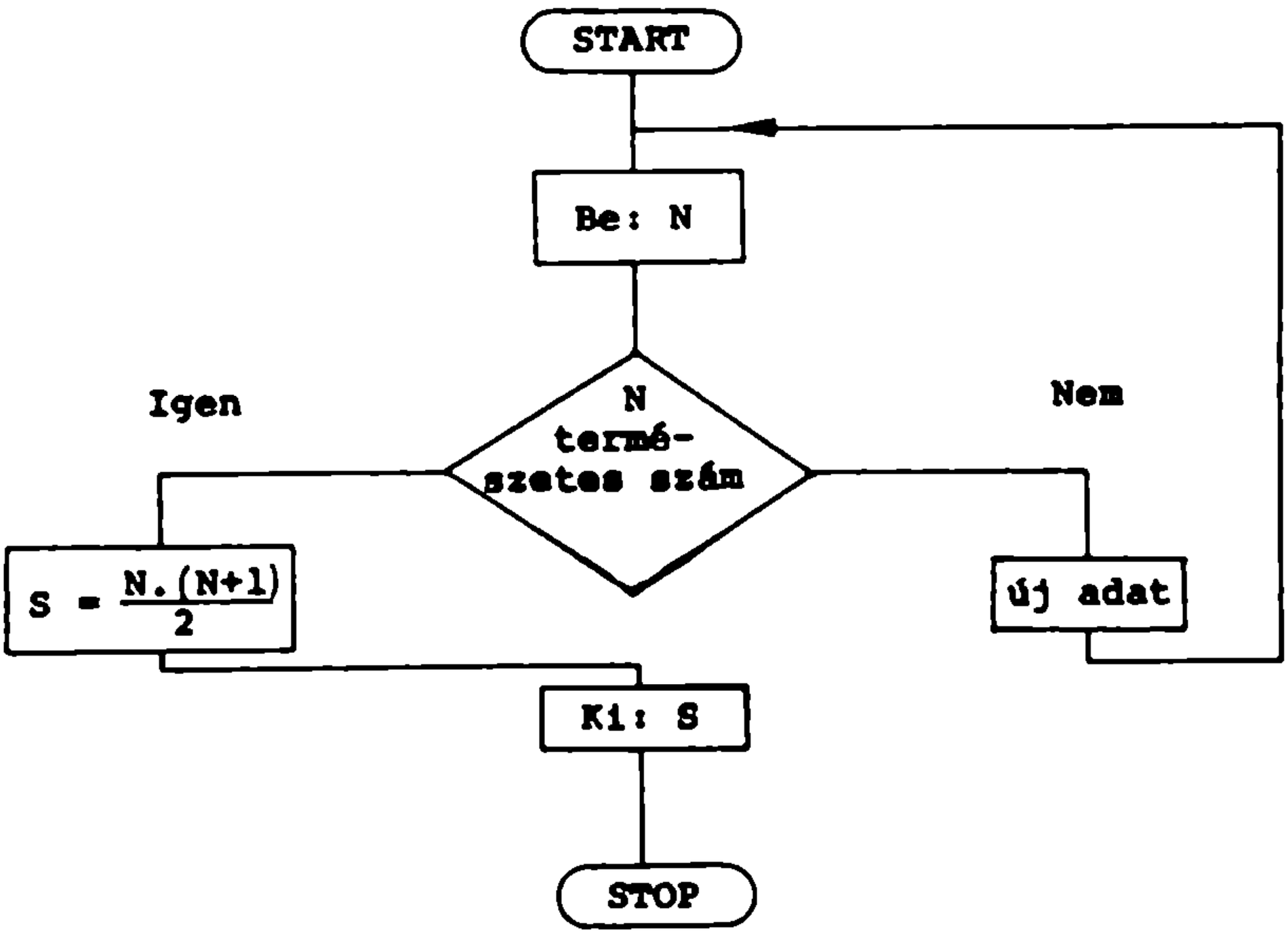
továbblépés három irányban is lehetséges, pl. az adott kifejezés kisebb mint nulla, nulla, vagy nagyobb mint nulla esetén. A lépések egymásutánját az összekötő nyilazott vonalak jelzik.

Szokás még az egész eljárás kezdő- és végpontját „START"-tal, ill. „STOP"-pal jelölni.

Természetesen ugyanazon feladat megoldásához többféle algoritmus is készíthető, a helyes eredményhez többféle úton is eljuthatunk. Most néhány közismert feladat algoritmusát és blokkdiagramját készítjük el. Amikor már megismertünk egy programnyelvet is, visszatérhetünk ezekhez a feladatokhoz, és megírhatjuk a kész, futtatható programot is.

Hasznos lenne, ha először mindenki önállóan próbálná elkészíteni a feladatot, és csak ezek után vetné össze az itt közölt megoldással. Remélhetőleg ezeknél ötletesebb, szebb megoldás is születik!

1. feladat (1. ábra)



1. ábra

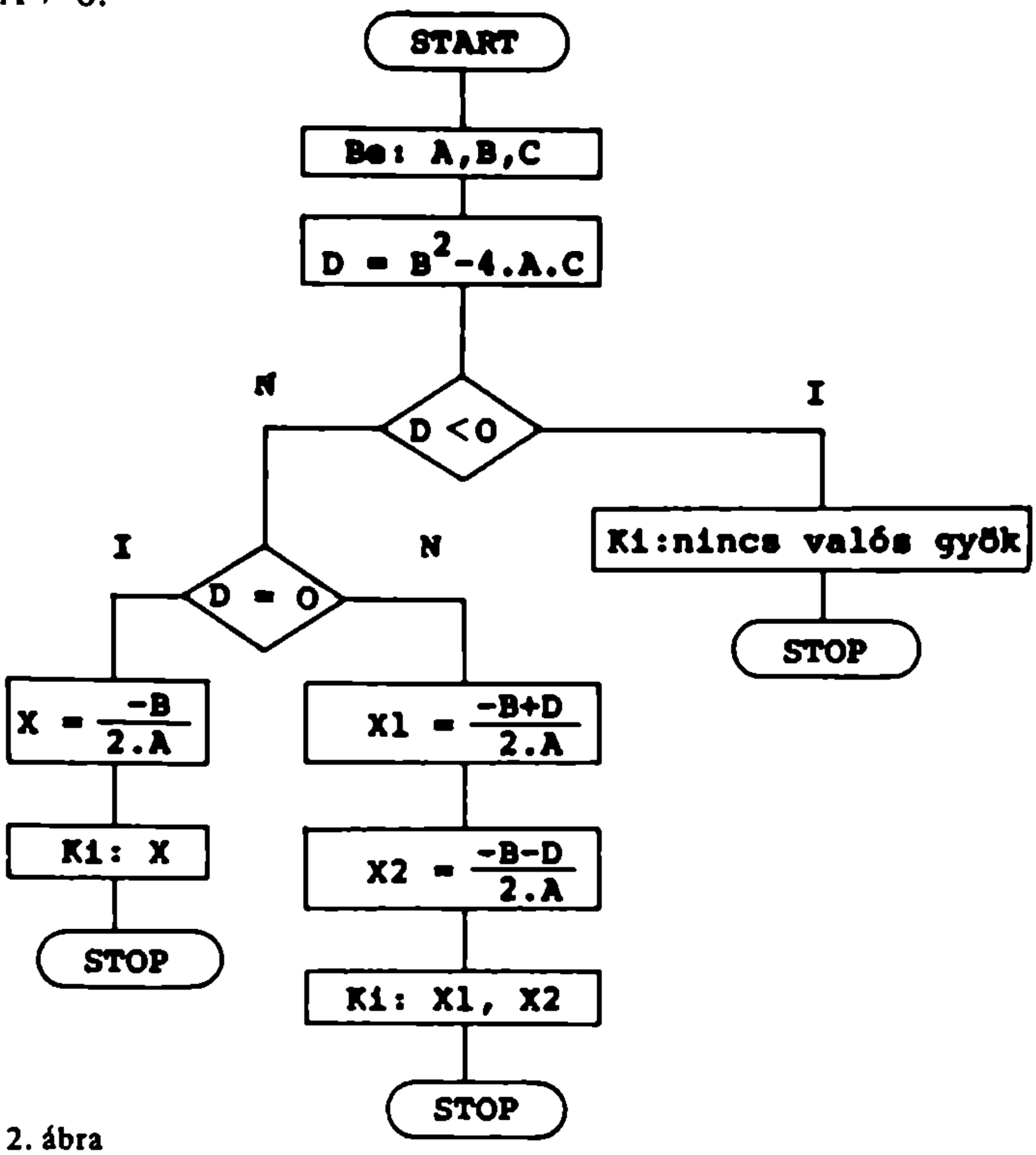
Készítsünk blokkdiagramot ahhoz a programhoz, amely kiszámolja az első n természetes szám összegét. Legyen n olyan bemenő adat, amelyet tetszőlegesen változtathatunk. Az összeget a következő képlet alapján számoltassuk ki:

$$S_n = N \cdot (N+1) \cdot \frac{1}{2}.$$

2. feladat (2. ábra)

Az $A \cdot x^2 + B \cdot x + C = 0$ másodfokú egyenlet megoldóképletének a blokkdiagramját készítsük el, feltételezve, hogy valóban másodfokú az egyenlet, tehát

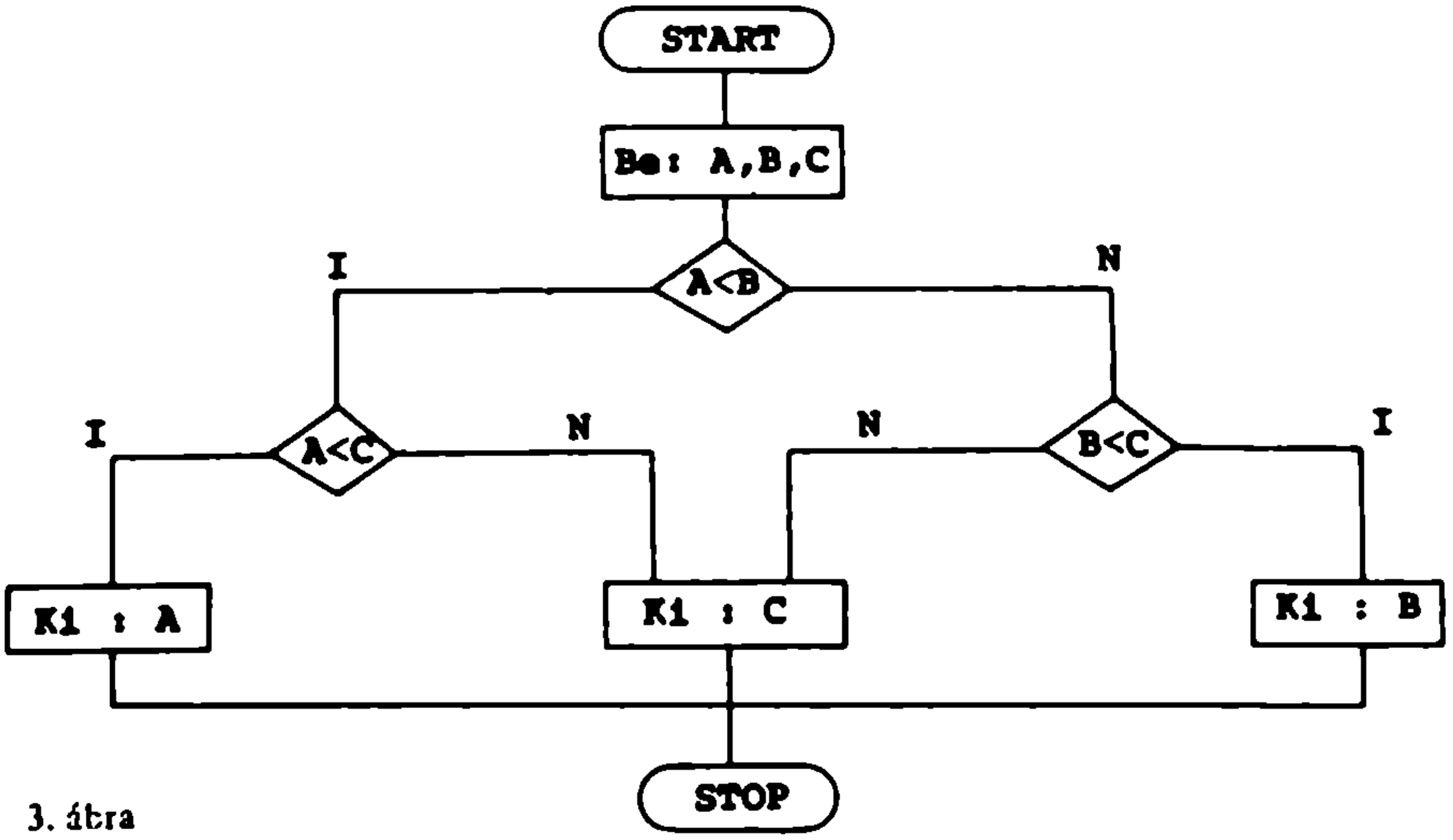
$A \neq 0$.



2. ábra

3. feladat (3. ábra)

Három különböző szám közül, amelyeket bemenő adatnak tekintünk, válassza ki a program a legkiscbbet, és azt írassuk ki!



3. ábra

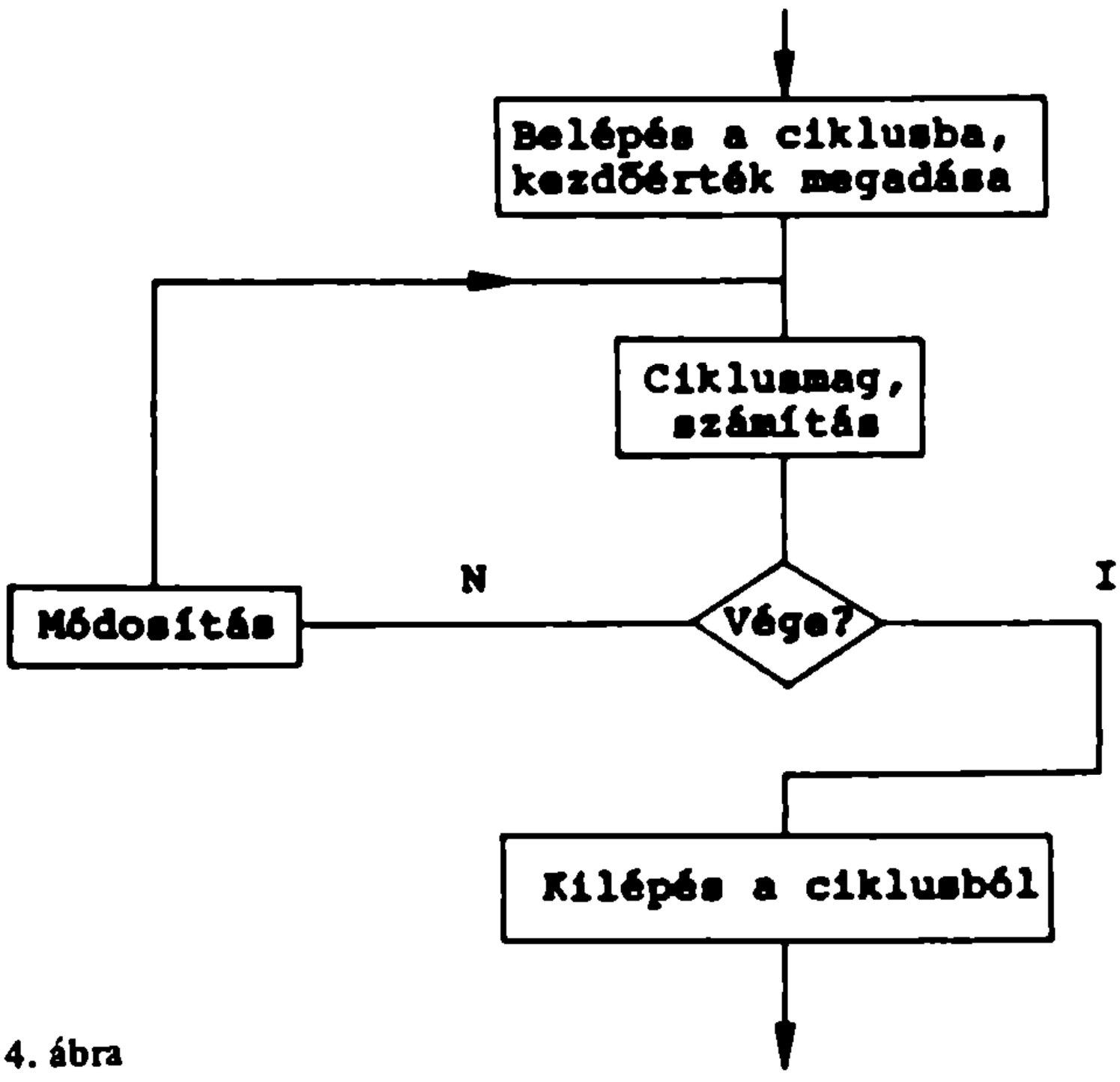
Megjegyzés: A döntési műveletek vizsgálatánál (szokás ezeket feltételes ugró utasításnak nevezni) a fő nehézség abból adódik, hogy meg kell vizsgálni minden lehetséges utat a továbblépéshez. Ezt figyelembe véve, a közölt algoritmus akkor működik igazán helyesen, ha az A, B, C értékek különbözőek. Finomítható a feladat, ha az esetleges egyenlőség esetét is tartalmazza.

Próbáljuk meg ezzel a kiegészítéssel is megoldani, esetleg úgy, hogy a három számot a program növekvő sorrendben rendezze!

Ciklusszervezés

A számítógépek széleskörben és nagyon kényelmesen használható képessége, hogy ugyanazt a műveletet vagy utasítás-sorozatot tetszőlegesen sokszor elvégeztethetjük, akár állandó, akár az eljárás során változó értéket fölvevő bemenő adatokkal. Ezt a módszert nevezzük ciklusképzésnek vagy ismétlődő (iterációs) cselekvésnek.

Az általános ciklusutasítás blokkdiagramja a 4. ábrán látható.



4. ábra

A „számítás" részt, amely során a ciklikusan elvégzett művelet zajlik, ciklusmagnak nevezzük. Ebben az esetben a „döntés-utasítás"-nak különleges szerepe van. Amennyiben hibásan vizsgáljuk meg továbblépés lehetőségeit, a számítógép végtelen ciklusba kerülhet, azaz a számításokat vég nélkül végzi, és az eredményt nem tudja felhasználni a további lépésekben.

Nagyon egyszerű példa a következő utasítás: 100 GO TO 100 , amely hatására a program tetszőlegesen sokáig „várakozik", és csak a futás megszakításával (BREAK) lehet kilépni a ciklusból.

Az ilyen döntés-vizsgálatoknak általában két fő típusa van:

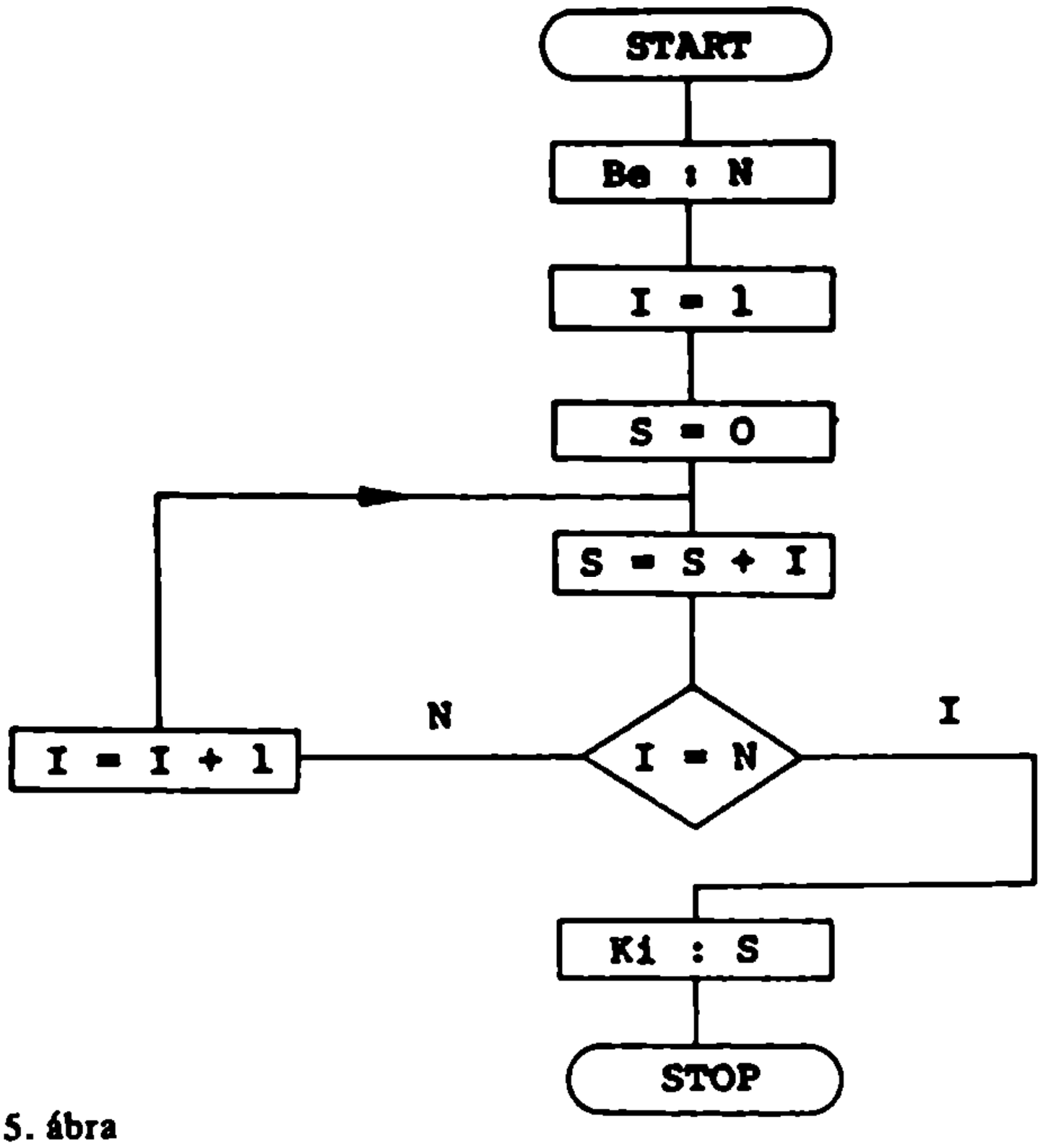
a) Számlálás, amelynél előre megadott számú ismétlés után történik meg a kilépés.

b) Egy olyan változó értékének a vizsgálata, amelyet a számítás során kapunk (pl. bizonyos pontosság vizsgálata stb.).

Természetesen a két lehetőség együtt is használható.

4. feladat (5. ábra)

Az első n pozitív egész szám összegének kiszámolása, az összegképlet felhasználása nélkül!



5. ábra

Akik először látnak olyan egyenlőséget, mint: I = I+1, bizonyára furcsának találják. A számítástechnikában az elfogadott és gyakran használatos jelölés, és nem azt a matematikai egyenletet takarja, amelynek nincs gyöke. Ez a szimbólum azt tartalmazza, hogy az I változó aktuális értéke legyen 1-gyel nagyobb, mint I előző értéke volt.

5. feladat (6. ábra)

Készítsük el annak a programnak az algoritmusát, amely kiszámolja egy tetszőleges N természetes számig az összes természetes szám szorzatát. A matematikában ezt a szorzatot N!-sal jelölik, és N faktoriálisnak mondjuk. Tehát N! = 1·2·. . . . . .(N–1)·N. Csak olyan számokkal dolgozzunk, amelyekre N! elfér a memóriában.

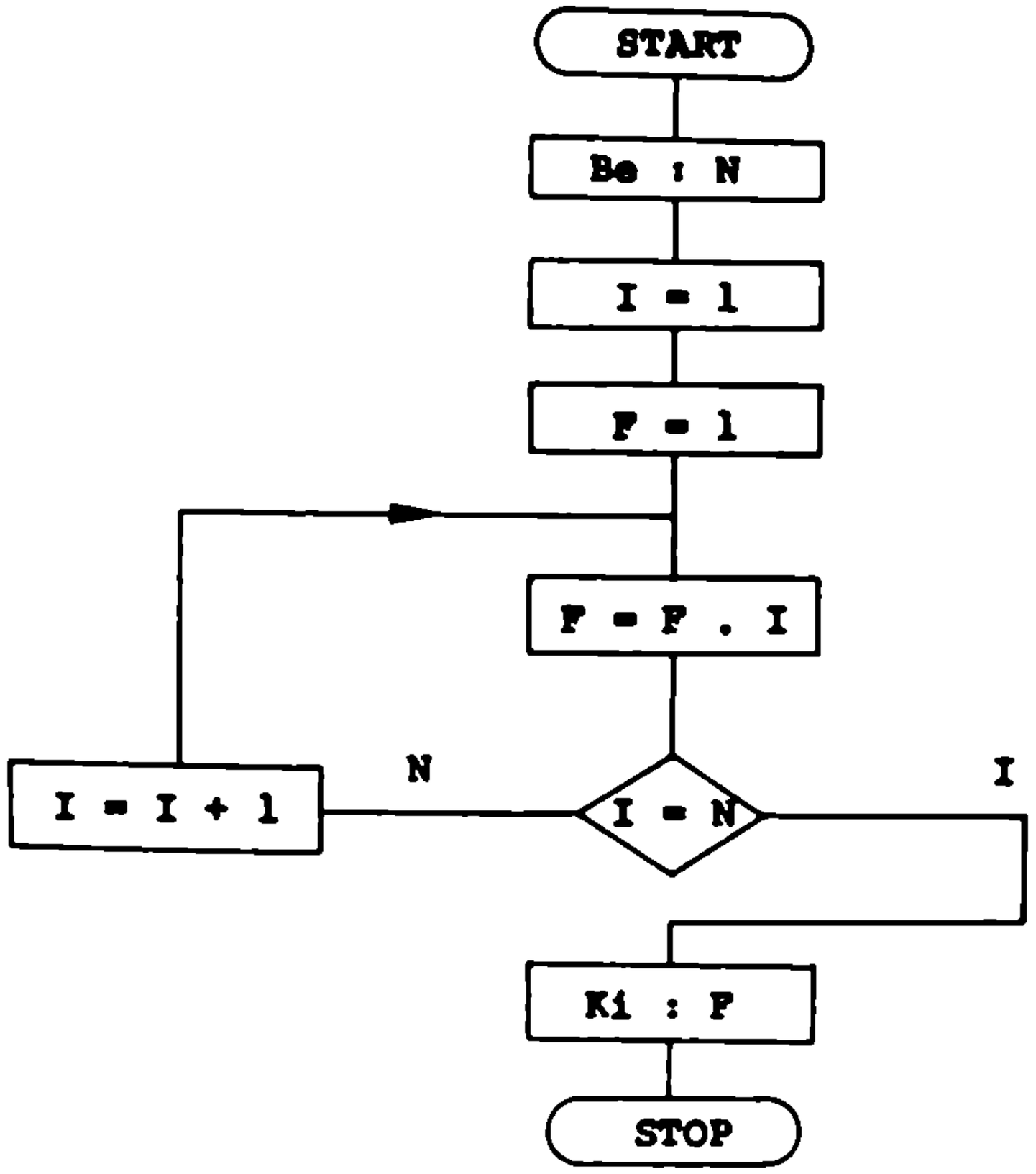
Nézzünk meg még egy érdekes matematika feladat talán nem egészen közismert megoldását!

6. feladat

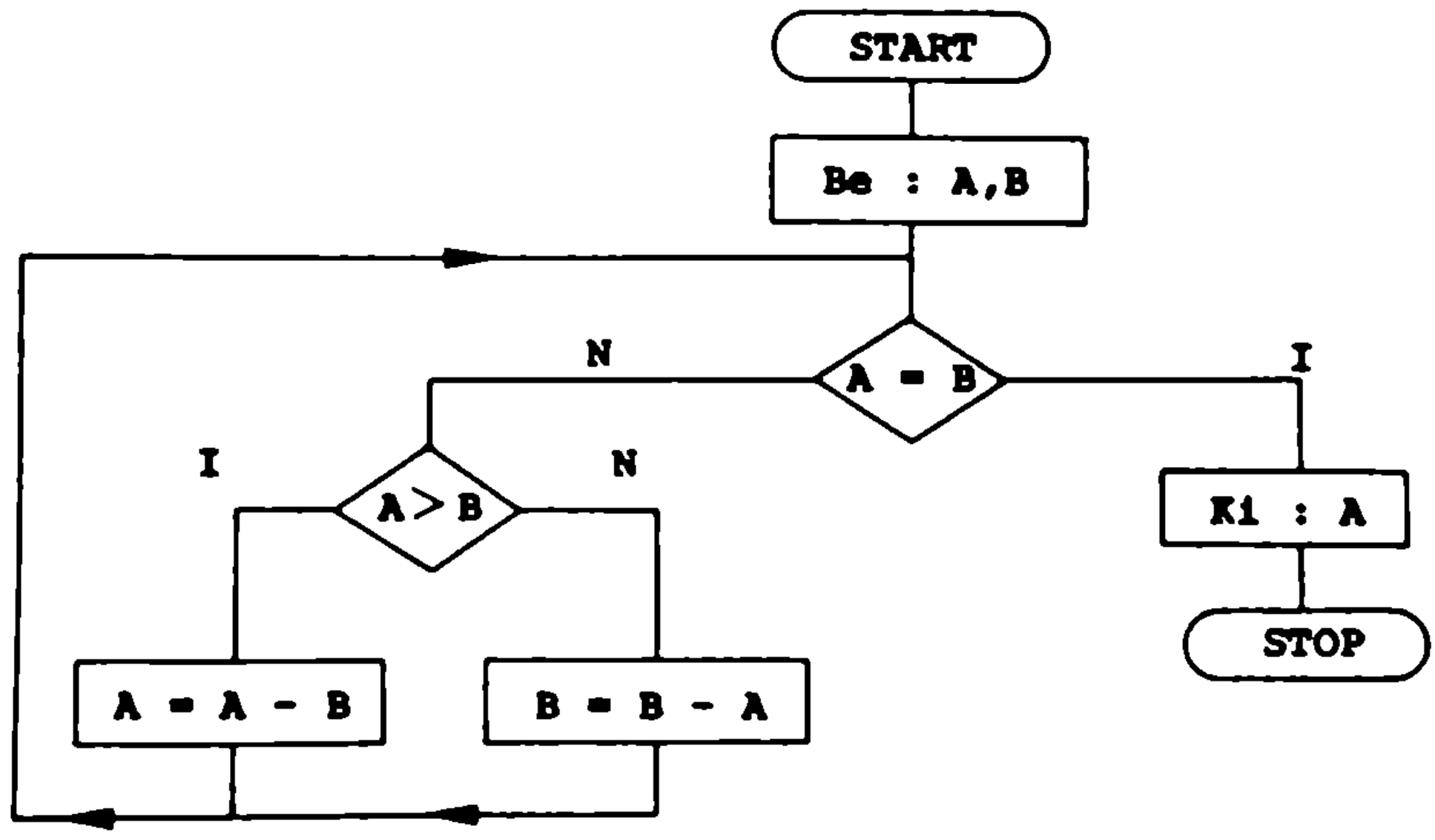
Írjunk algoritmust két pozitív egész szám legnagyobb közös osztójának a meghatározására. A matematikában több módszer ismeretes erre a problémára, és ezek az eljárások többnyire nagyon jól algoritmizálhatók.

A bemutatott megoldás előnye az, hogy a gép aritmatikai egységét alig-alig „dolgoztatja". Az alapgondolat az, hogy ha egy S szám osztója A-nak és B-nek is, akkor osztója a két szám különbségének is. Ez a különbség mindig kisebb, mint A és B közül a nagyobbik, legyen ez pl. A. Ezután B-t és A–B-t vesszük az eredeti két szám helyett, és ismételjük az előbbi eljárást mindaddig, amíg a két szám egyenlő nem lesz. Belátható és bizonyítható, hogy ez a végső érték lesz az eredeti két szám legnagyobb közös osztója.

Az algoritmus véges sok lépésen belül eredményre vezet. Ezt nem bizonyítjuk, de gondoljuk végig! A programot elkészíthetjük pl. a következő eljárás segítségével (7. ábra).



6. ábra



7. ábra

7. feladat

Végül nézzünk egy olyan problémát, amely nem közvetlenül matematikai jellegű, azaz megoldásához a matematikai modellt kell elõször kitalálnunk. Ez a feladat az adattömbök kezeléséhez lesz jó példa!

Tulajdonképpen ezért is érdemes hasonló jellegű „játékokkal" foglalkozni már a számítástechnikai ismereteink kezdetén, hogy nehogy valaki abba a hibába essen, hogy „programozni" csak matematikusok tudnak, és csak matematikai feladatokat!

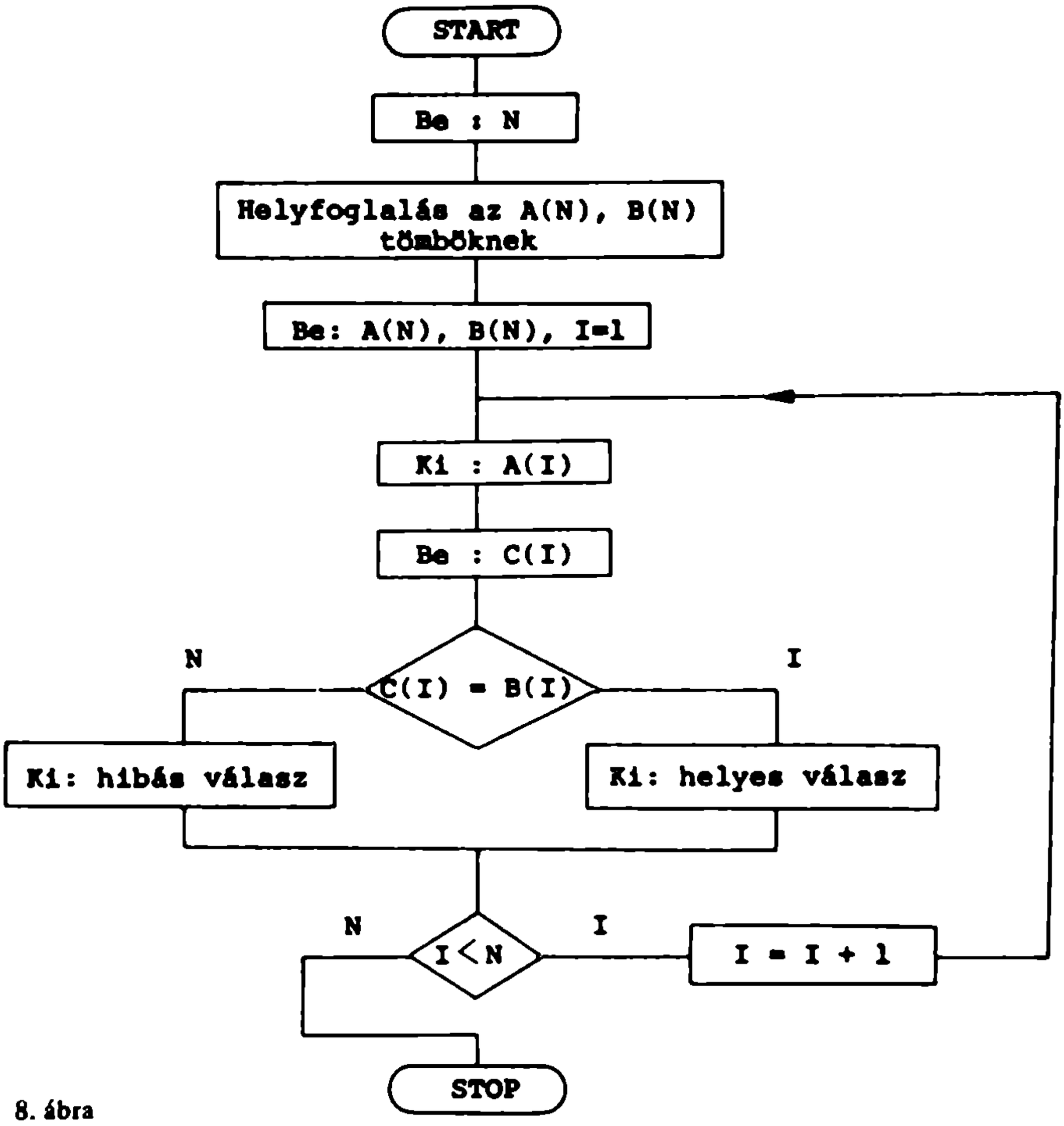
Ez a feladat arra szolgál, hogy pl. idegen szavak magyar megfelelõit visszakérdezzük. Nevezhetjük akár „szódolgozatnak" is! A két csoportban beírt adatokat, azaz az egyikben

az idegen szavak, A(N), a másikban a magyar jelentésük, B(N), – „a másik játékos" ezt nem ismeri.

Az egyik csoport elemeit egyesével kiíratjuk: A(I), a „felelő" pedig beírja a szavak megfelelőit, ha ismeri: C(I).

Amennyiben a C(N) tömb elemei rendre megegyeznek a B(N) tömb megfelelő elemeivel, akkor „helyes" a „felelő" megoldása, ha pedig nem, akkor „hibás".

A program (8. ábra) természetesen nem kizárólag idegen szavak „kikérdezésére" szolgál, használható bármilyen két halmaz elemeinek az összehasonlítására.



8. ábra

Különösen hasznos, amíg nem rendelkezik valaki nagy programozási gyakorlattal, hogy a program megírása előtt elkészítse a feladat folyamatábráját. Így sokkal nagyobb eséllyel találja meg az optimális utat, és amennyiben a program hibás, annak okát is könnyebb lesz kinyomozni!

A SZEMÉLYI SZÁMÍTÓGÉPEK KIALAKULÁSA ÉS MÜKÖDÉSE

Az egész világon, így hazánkban is, akik programozni akarnak, függetlenül a feladattól – matematikai, gazdasági jellegü vagy játék –, a legkönnyebben a személyi számítógépekhez tudnak hozzájutni. Ezek felhasználási területe rendkívül széleskörü, népszerüségük oka összetett: emberközelség, ár, hozzáférhetőség, egyszerüség. Meg kell azonban jegyezni, hogy memóriakapacitásban nem veszik föl a versenyt a nagyobb számítógépekkel.

A személyi számítógép egy „kicsi, olcsó, általános célú digitális számítórendszer", amely központi egységként egy LSI (Larga Scale Integrated – nagymértékben integrált) mikroprocesszor-elemet tartalmaz, és valamilyen magasszintü programozási nyelven irt (BASIC, PASCAL) programok tárolására és végrehajtására alkalmas. Az információk általában irógépszerü billentyüzetről irhatók be, a kimeneti jel pedig a géphez csatlakozó televízió képernyőjén jelenik meg. Szabályos hangfrekvenciás mágnesszalagos kazetta használatával a billentyüzettel bevitt programok és adatok rögzíthetők és tárolhatók, így nem szükséges minden használatkor begépelni azokat. A nálunk forgalomben levő számítógépek nagy részénél Z80-as típusú, 8 bites mikroprocesszort használnak központi egységként. A felhasználó részére géptípustól függően különböző pl. 16 kbájt, 48 kbájt vagy 64 kbájt szabad tárterület áll a programok és az adatok rendelkezésére. A memóriának van olyan területe, amelynek tartalmát a programozás során nem szabad elveszíteni vagy megváltoztatni. Itt kap helyet pl. a BASIC nyelv értelmező programja, amelyet már a gyártás során építenek be a tárba. Erre a gazdasági megfontolásokon kívül azért is szükség van, mert ha minden programot kazettáról töltenénk be, lényegesen lassúbb és instabilabb lenne a rendszer. Hátránya viszont az, hogy ha már egyszer tartalmazza a BASIC értelmezőprogramot, akkor ezen már változtatni nem lehet, más programozási nyelv alkalmazása szinte megoldhatatlan. Igaz ugyan, hogy pl. egy egyszerübb PASCAL nyelvet értelmező program 10–12 kbájton elfér, és így maradhat a program számára is elegendő memória.

A BASIC nyelv jelen pillanatban egyértelmüen a személyi számítógépek „sikernyelve"! Az egyszerüség, a könnyen tanulhatóság mellett hallatlan nagy előnye, hogy gyakorlatilag az összes matematikai és adatkezelési függvényt tartalmazza, amelyre a programozás során szükségünk lehet. Bizonyos esetekben korlátozza a használhatóságát a lassú futás. A programot ui. nem a futtatásra kész, a gép számára érthető utasítások alkotják, hanem egy olyan karaktersorozat, az ún. ASCII (American Standerd Code for Information Exchange) karakterek, amelyet az értelmező programnak soronként meg kell vizsgálnia és a megfelelő kódú gépi rutint aktivizálnia.

Persze ez a lassúság nem behozhatatlan hátrány, a BASIC nyelv egyéb előnyei bőven kárpótolják érte a programozót. Így pl. a hibakeresés, a hibajavítás sokkal egyszerübb és gyorsabb, mintha egy másik, magas szintü nyelven bevitt programban kellene, a fordítás után megtalálni a hibát. Ennél komolyabb problémát jelent az a tény, hogy a BASIC nyelven megírt programnak és a BASIC-et értelmező programnak egyszerre kell a tár megfelelő helyén lenni, és ez a felhasználó számára nagyon lecsökkenti a szabad tárterületet. Például az a „rutinprogram", amely a billentyüzet használatát biztosítja, minimum 100 programsort vesz igénybe. Az is igaz ugyan, hogy ma amikor a viszonylag drága ferritgyürüs memóriákat a sokkal modernebb és olcsóbb félvezetőtárak váltják fel, a nagy tárigény sem jelent megoldhatatlan gondot a tervezőknek. A szükséges tárterület egyrészt a használt mikroprocesszortól, másrészt a felhasznált BASIC-tól függ. A legkisebb tárigénye

az ún. TINY-BASIC-nek van, mindössze 2 bájt. Ez a BASIC változat csak a −32768 és +32767 közé eső egész számokkal tud dolgozni. Egy „átlagos" igényt kielégítõ BASIC kb. 8 kbájton fér el, 16 kbájton pedig már speciális problémák is programozhatók: grafikus funkciók, hanggenerátor.

Ha a számítógépünkhöz egyéb periférikus egységek is csatlakoznak, mint pl. sornyomtató, mágneslemez stb., akkor ezek használatához a kiterjesztett BASIC értelmezõ program szükséges, amely 45 kbájtot igényel. (Megjegyzés: számítógép vásárlás esetén érdemes ügyelni ezekre a szempontokra.)

Az eddig elmondott funkciókat a mikroszámítógépek meglehetõsen bonyolult rendszerként képesek megoldani, pedig tulajdonképpen csak néhány integrált áramköri elemet tartalmaznak. Ha kiindulunk egy kicsit leegyszerûsített „számítógép-modellbõl", akkor ennek az összetett rendszernek a mûködése minden bizonnyal érthetõbb lesz. Márpedig olvasóink közül valószínûleg többnek is elõbb vagy utóbb megfordul a fejében az a gondolat, hogy akár „építeni" is lehetne egy mikroszámítógépet. A technológia hallatlanul gyors fejlõdését figyelembe véve, erre nálunk is lehetõsége van az érdeklõdõknek.

Ez a bizonyos „alapgép" csak két nagy egységet tartalmaz: a központi tárat és a központi egységet.

Központi tár

Itt található valamennyi felhasznált és feldolgozandó információ, egységes formában kódolva, 1 bájt hosszúságú rekeszekben. (Megjegyzés: a jelenleg legelterjedtebb mikroprocesszort, a 8 bitest vesszük alapul.) A központi tár rekeszei és a központi egység átmeneti tároló regiszterei között történik a programfutás alatt az információk és az adatok áramlása, cseréje.

A tár rekeszei címezhetõk, és a megfelelõ címû rekesz kijelölése a címregiszterben történik. A címregiszter általában két bájt hosszúságú, mivel így a rendelkezésre álló 16 biten $2^{16} = 65536$ tárrekesz címezhetõ. Ez a 65 kbájt (pontosan 65·1024 bit) már elegendõ a forgalomban levõ személyi számítógépek tárcímzéséhez.

A központi – vagy operatív tár – kétféle tártípusból állhat, mindkettõ korszerû, magas integráltságú félvezetõ típusú tár:

1. Program tárolásra használt tár

Az itt tárolt információk a tápfeszültségtõl függetlenül mindig bent vannak a számítógépben, bekapcsoláskor azonnal aktivizálhatók. Az ilyen típusú tárnak is több változata ismeretes.

a) ROM (Read Only Memory): csak olvasható tár. A feltöltése a számítógép kialakítása során történik, az állandóan használt programokat, rutinprogramokat tartalmazza (pl. BASIC értelmezõ program), a programozó által nem módosítható és hozzáférhetõ.

b) PROM (Programmable Read Only Memory): programozható, de csak olvasható tár. Bizonyos eljárással utólag is programozható, de a beírt program már nem módosítható, hibás beírás esetén a tár használhatatlanná válik.

c) REPROM (Re-Programmable Rom): újraprogramozható, de csak olvasható tár. Az itt tárolt információk hibás beírás esetén vagy módosításkor ultraibolya fénnyel törölhetõk vagy újra írhatók, mindez azonban igen lassú és költséges eljárás.

Léteznek még az ún. EAROM (Electrically Alterable ROM) tárak, amelyek annyiban egyszerûbbek, hogy a törlés és az újraírás is elektromos jellel történik.

2. Adatok tárolására használt tár

Egyaránt alkalmas az információk gyors kiolvasására, ill. a beírásra, de tartalma „elveszik"

a tápfeszültség kikapcsolásakor. Sőt még üzem közben is, kb. 10 ms-onként egy külön „felfrissítő" áramkör felhasználásával stabilizálni kell a tartalmat. Ezek az ún. dinamikus-RAM tárak: (Random Access Memory) tetszőleges hozzáférésű vagy írható-olvasható tár. A csekély előállítási és szerelési költségek és egyéb előnyös tulajdonásgok miatt a mikroszámítógépekben elterjedten használják.
Egy adat „hozzáférési ideje" 100. . .400 ms.

A statikus RAM tárakat tulajdonképpen a mikroszámítógépekhez fejlesztették ki, a legkorszerűbb CMOS (olyan félvezető technológia, amelynek a teljesítmény felvétele egészen minimális) technológia felhasználásával. A közös tulajdonságok mellett annyiból jobb, hogy nincs szükség felfrissítésre.

Központi egység

Közismert elnevezése a CPU (Central Processing Unit). A számítógép „magja", itt történik a vezérlés, a műveletvégzés. Itt kapnak helyet a regiszterek, amelyeknek a száma és funkciója meglehetősen típusfüggő. Az általános célú regiszterek a következők:

a) Utasításszámláló (PC: Program Counter) regiszter. Ez tartalmazza a soron következő utasítás – amely a tár egy adott rekeszében van elhelyezve – címét. Tekintettel arra, hogy az operatív tár címregisztere 16 bites, így az utasításszámlálóban is 16 bit hosszúságú információkat kell elhelyezni. A regiszter tartalma minden végrehajtott lépés után 1-gyel nő, ha pedig ugró utasításhoz ér a program, a továbbszámlálás helyett új értéket vesz föl.

b) Akkumulátor (AC) regiszter. Ha egyáltalán létezik fontossági sorrend, akkor abban ez az első! Az elvégzendő művelet egyik operandusát tartalmazza, és átmenetileg az eredményt. Tárolja a műveletvégzés során esetleg keletkező túlcsordulást is, és az átvitel bitet. E regiszter által tárolt bitek számát mindig az alkalmazott mikroprocesszor szóhossza határozza meg. Tehát a 8 bites mikroprocesszorok esetében ez a regiszter is 8 bites.

c) Utasítás regiszter (IR: Instruction Register). Ez is 8 bites, itt kap helyet az éppen végrehajtás alatt levő utasítás kódja.

d) Vezérlőegység: Azokat a jeleket állítja elő, amelyek nyitják, ill. zárják az információk útját az operatív tár és a központi egység regiszterei között.

e) Aritmetikai Logikai Egység (ALU: Aritmetic Logic Unit). Egy olyan kombinációs hálózat, amely az adatokból a kívánt eredményt képezi, amelynek eredménye az akkumulátorba kerül, törölve annak előző tartalmát. Működése, felépítése függ a gép kialakításától.

A központi egység és a központi tár együttes, összehangolt működése előfeltétele a hatékony munkának. Még a legelemibb művelet, egy összeadás elvégzése is többszörös oda-vissza kapcsolatot – gépi ciklust – jelent ezek között az egységek között.

Szólni kell még röviden a személyi számítógépek periférikus egységeiről, amelyek itt sokkal jobban összefonódnak elhelyezés és működés szempontjából egyaránt a központi géppel, mint a nagy számítógépeknél. Ezek a következők:

1. Billentyűzet. Ez a minimálisan szükséges beviteli eszköz ahhoz, hogy valamilyen magas szintű programnyelven kommunikálni tudjunk a számítógéppel. A használatos karakterkészlet az ún. ASCII kód alapján van kódolva.

2. Kazettás mágnesszalagos tároló. Ez egy közönséges hangfrekvenciás magnetofon, amely a programot és az adatokat mágneses jelekként képes tárolni.

3. Képernyős megjelenítő, amely a bevitt utasításokat, adatokat és a kiírt eredményeket megjeleníti a felhasználó előtt. Lehet egyszerűen egy TV készülék, vagy a legegyszerűbb esetben egy folyadékkristályos kijelző.

4. Nyomtató. Vannak amelyek a hagyományos távírógéphez hasonló sornyomtatók, és léteznek saját mikroprocesszort tartalmazó gyorsnyomtatók is. A kiírandó információkat általában egy elektromos jelre érzékeny, alumíniummal bevont papírra viszik föl.

5. Hajlékony mágneslemezes rendszer, amely minden szempontból a legjobb perifférikus gyorstároló. A hajlékony mágneslemez vagy más néven flopy, egy vékony rugalmas plasztiktárcsa, amely mágneses réteggel van bevonva. A mágnesszalaggal szemben nem „soros hozzáférésű", tehát nem kell végigpörgetni az egész lemezt ahhoz, hogy egy adott helyen levő információt betöltsük a gépbe. A tartalom betöltése rendkívül gyors, a tárolás megbízható, és egy-egy lemez akár 200 kbájt adatot is képes tárolni.

Természetesen a személyi számítógépek az „extra" egységek nélkül, alapkiépítésben is jól használhatók a legkülönbözőbb feladatokhoz. Azonban a fejlődés tendenciáját tekintve, a közeljövőben ezek az elemek is ugyanolyan elterjedtek és hozzáférhetőek lesznek, mint maga a gép.

Nálunk az 1980-as évek kezdetétől terjednek mind szélesebb körben és nagyobb számban a személyi számítógépek. A „nagy ugrás" az 1983-as év volt, amikor egységesen, valamennyi középiskolában hozzáférhetővé vált a hazai gyártmányú és szervizelésű „Iskolai számítógép".

Álljanak itt felsorolásszerűen, figyelembe véve a „kínálat" szinte napi változását, a jelenleg forgalomban levő és legelterjedtebb személyi számítógépek típusai: ABC 80, HT 1080, HT 2080, ZX 81, ZX SPECTRUM, AIRCOMP, COMMODORE, PRIMO.

A PROGRAMOZÁS KÜLÖNBÖZŐ SZINTJEI

Programozási nyelvnek azt a jelölésrendszert nevezzük, amely felhasználásával a feladat végrehajtásához szükséges utasítások sorozatát, azaz magát a programot leírtuk. A legegyszerűbb esetben mindez tulajdonképpen egy bináris számsorozat, amely a gép kiépítése alapján a gépi utasítások kódjait tartalmazza. Ha ezen számkódok betűképeit vihetjük be, a mi dolgunk egyszerűbb, a gép munkája viszont lényegesen komplikáltabb.

A programnyelveknek három fajtáját szokás megkülönböztetni:

1. Gépi kód.
2. Szimbolikus nyelv vagy Assembly.
3. Magas szintű programnyelv.

Nézzük meg közelebbről az „idegen nyelveket" általánosan, és azután a lehetőségekhez képest teljes részletességgel egyet. Nem azért választottuk ki a BASIC nyelvet, az a „legjobb"! Nincs legjobb programnyelv! Azért döntöttünk így, mert jelenleg, 1984-ben, és még minden bizonnyal egy ideig ez lesz az a programnyelv, amelyen keresztül nálunk a legtöbben meg tudnak ismerkedni ezzel a tudománnyal.

1. A gépi kód.

A számítógép csak ilyen utasításokat képes értelmezni, és végrehajtás után válaszolni azokra. Ez a számítógép „anyanyelve". A kezdeti időszakban, az egyetlen lehetőség ez volt a programozásra. Ma viszont a számítógépek többségénél a felhasználó alig tud gépi nyelven dolgozni, helyette „emberi nyelven" beszélget a géppel. A számítógép működése viszont sokkal érthetőbb, sokkal inkább közel kerül hozzánk ha ezt a lehetőséget is megismerjük, megjegyezve, hogy a gépi nyelven történő programozás nehézkes, hosszú munka, és nem is vezet okvetlenül rövidebb futási időhöz!

A gépi kódú programozás főbb elvei és egyben nehézségei is a következők:

a) A műveleti kódokat numerikus formában kell megadni, általában hexadecimális alakban.

b) Az egyes műveleteket „abszolút" elemi szintre kell lebontani.

c) Ismerni kell a központi tár pontos felosztását, és csak adott tárterületre történhet a program betöltése.

A gépi kódú programozás nagy hátránya az, hogy az esetleges hibáról futás közben semmiféle információt sem szerezhetünk, még csak a hiba jellegéről sem. Ezt nekünk kell kideríteni. Természetesen bizonyos feladatokhoz megéri vállalni ezeket a nehézségeket, pl. a rövidebb futási idő vagy a memória ideális kihasználása érdekében.

Az elmondottak illusztrálására nézzük meg egy nagyon egyszerű feladat gépi nyelven írt programját a HT 1080 iskolai számítógépre, amelyben a központi feldolgozó egység egy *Z 80*-as mikroprocesszor, amely eléggé elterjedt ahhoz, hogy a program hasonló legyen több más személyi számítógépben is. Két egész szám összeadásáról van szó. Legyen az A regiszterben 15, a B regiszterben pedig 30, a két szám összegét pedig kérjük az A regiszterben:

A program az adatokkal együtt:

binárisan	hexadecimálisan
1. 0011111000001111	3E 0F
2. 0000011000011110	06 1E
3. 1000000000101101	80 2D

Az első lépésben az A regiszterbe betesszük az egyik számot, a második lépésben a B regiszterbe a másikat, a harmadik lépésben pedig az A regiszterbe a két regiszter tartalmát. A programozónak meg kellett, és ma is meg kell tanulnia a gépi szintű utasítások kódjait, ha így akar kommunikálni a géppel.

A nehézségek áthidalását próbálták megoldani a gépi nyelvhez még közelálló, de a gépi műveletek számkódjait különböző kulcsszavakkal helyettesítő, szimbolikus nyelvekkel. Ez az Assembly nyelv.

2. A szimbolikus nyelv.

Az Assembly még mindig gépfüggő, tehát egy program csak egy gépen futhat, és ehhez is szükséges a gépi nyelvre lefordító fordítóprogram (assembler editor). Az assembly szintű utasítások a tényleges gépi utasításoknak felelnek meg, de nem egy „száraz" számhalmazként, hanem szemléletesebb, könnyebben felismerhető és áttekinthető formában.

Az assembler utasításai három részre tagozódnak, három mezőben írhatók le:

cím utasításkód operandus

A cím részbe tetszőleges betűsorozat írható, de maradhat akár üresen is. Akkor van jelentősége, ha a program során – pl. feltételes ugrás – hivatkozni akarunk egy adott utasításra.

Az utasításkód helyén tulajdonképpen az adott assembly nyelv alapszavai állhatnak. Ezek az adott utasítást leíró angol szó rövidítései, amely megfelel az adott gépi kódú utasításnak. Például:

LD: töltsd
ADD: összegezd
SUB: szorzás
JP: ugrás

Az operandus határozza meg azt a címet, amely címen levő adattal az adott utasítást végre kell hajtani. Ez lehet ún. „direkt" operandus is, amikor nem a memória egy címét, hanem közvetlenül annak tartalmát adjuk meg. Természetesen hivatkozhatunk más a programban szereplő címre is.

Például:

LD A,10: az A regiszterbe elhelyezzük a 10 számot.

LD A,B: az A regiszterbe tároljuk a B regiszter tartalmát.

LD A, (32000): az A regiszterbe tároljuk a memória 32000-es című rekeszének tartalmát.

ADD A,B: az A regiszterbe elhelyezzük a B és az A regiszterek tartalmának összegét. Ezek alapján az előzőekben említett egyszerű összeadási példa Z80-as assembler programja a HT 1080 gépre a következő lesz:

1. LD A,15 (A-ba 15)
2. LD B,30 (B-be 30)
3. ADD A,B (A-ba az összeg)

A gépi nyelv és a szimbolikus összehasonlításakor látható a szimbolikus nyelv előnye. Azonban még ez is magán viseli a gépi kód jellegzetességeit. Például minden gépi utasításnak megfelel egy-egy assembler utasítás. Ebből következik, hogy ez a nyelv is erősen géphez kötött. Ezen kívül hosszú és bonyolult szerkezetű programok megírása hosszadalmas és nehézkes. A programozói munka megkönnyítését, hatékonyabbá tételét szolgálják az ún. ROM-rutinok, amelyek felhasználásával bizonyos gyakran használatos feladatok megtervezését nem kell assembly szinten elvégezni. Ilyen pl. az aritmetikai rutin, a grafikus rutin stb.

Megjegyzés: a HT 1080 gépen a gépi kódban írt programok (pl. a játék-programok többsége) betöltése különbözik a BASIC programokétól. A LOAD helyett a SYSTEM paranccsal indíthatjuk a beolvasást, majd a képernyőn megjelenő „?"-re az adott program nevét kell megadni. Ezután indítható a magnetofonról a beolvasás.

Ismeretes több olyan programnyelv, amelyek az assembly nyelvekből fejlődtek ki, és bizonyos géptípusra ill. felhasználási területre dolgozták ki. Ezek az „Autokód"-ok. Mivel nem nyújtottak a programozók számára hatásos új eszközöket, nem terjedtek el széles körben, de ma is használatosak.

3. Magas szintű programnyelv.

A gépfüggetlen, univerzális nyelvek kialakítása egyszerűsítette le lényegesen a felhasználói munkát. Az így megírt program minden olyan számítógépen lefuttatható, amely rendelkezik az adott nyelv fordítóprogramjával (ez a compiler). Így nem szükséges minden feladatot minden gépen újra programozni, használhatók a „programkönyvtárak" is.

A legelterjedtebb, a lehetőségekhez képest gépfüggetlen, feladatorientált, de általános jellegű programnyelvek a következők:

Fortran

A Formula Translation szavak rövidítése. Különösen a tudományos, matematikai jellegű feladatok programozására alkalmas. A FORTRAN az assembly utasításokhoz képest két fontos új elemet tartalmaz:

a) Az utasításokban tetszőleges aritmetikai formulákat használhatunk. Például az $X = B^2 - 4 \cdot AC$ képlet FORTRAN utasítása a következő:

X = SORT(B2–4AC)

b) Lehetőség van a többször használt programrészek hívására (szubrutin használat).

A FORTRAN nyelv nagy előnye, hogy könnyen és gyorsan elsajátítható. Hátránya, hogy nem rendelkezik azokkal az eszközökkel, amelyek a jól tagolt, világosan átlátható szerkezetű programok írását lehetővé teszik.

Algol

Az Algorithmic Language szavakból származnak. A összetett műszaki és tudományos feladatok megoldására dolgozták ki, a lokalitás elve jelenti az újdonságot az előző programnyelvekhez képest. A programrészek vagy „blokkok" tetszés szerint egymásba skatulyázhatók, és egy-egy blokkon belül saját programváltozók használható, anélkül, hogy ez zavarná a program többi részét. Ez a koncepció olyan sikeresnek bizonyult, hogy azóta is szinte minden magas szintű programozási nyelv alapját jelenti. Ma is az ALGOL nyelv továbbfejlesztett változatai adják a programnyelvek zömét.

Cobol

A Common Business Oriented Language angol kifejezés kezdőbetűiből származik a programnyelv elnevezése. Az olyan problémák megoldásához használható gazdaságosan, ahol nem a matematikai műveletek dominálnak, hanem a nagy tömegű adatfeldolgozási tevékenység. Ezért elsősorban a gazdasági életben, a közgazdasági jellegű feladatok programozására alkalmas.

PL 1 (Programme Language 1)

Az egész nyelv struktúrája azzal a céllal készült, hogy mint egy univerzális programnyelv egyaránt jól felhasználható legyen a tudományos, műszaki és adatfeldolgozási munkában.

Használatos még számos más, különböző mértékben elterjedt programnyelv, mint pl.: Pascal, Elan, Forth, APL stb. Ezek a nyelvek, bár közismert a széleskörű felhasználásuk, nem tartoznak könyvünk témájához.

A további részekben a BASIC nyelvről igyekszünk a lehetőségekhez képest részletes és teljes képet adni, mivel jelenleg hazánkban, középiskolai szinten ez a legelterjedtebb, legismertebb programnyelv.

BASIC

Ez is egy magas szintű programnyelv, a Begginer's All Purpose Symbolic Instruction Code – magyarul: „Kezdők általános célú szimbolikus utasításkódja" –, kifejezés rövidítése. 1964-ben, egy magyar származású tudós, John Kemény és kollégái fejlesztet-

ték ki Amerikában, kezdő programozók számára, de általános feladatok megoldására. Az alap a FORTRAN nyelv volt, és elsősorban a személyi számítógépek programozására tervezték. A programozó közvetlen kapcsolatban áll a géppel, billentyűzeten keresztül írhatók be az utasítások, és így akár futás közben is javítható, kiegészíthető a program. Az ismertebb személyi számítógépek mind „értik" a BASIC nyelvet, azaz rendelkeznek BASIC értelmező programmal, de a „beszélt" nyelvekhez hasonlóan mind csak a saját „tájszólásában". Nem valószínű, hogy pl. egy Sinclair Spectrum-ra írt program működőképes a HT típusú iskolaszámítógépen is! Mivel a számítógépeket gyártó cégeknek nagyon kényelmes, hogy a BASIC elég jól „igazítható" a különböző műszaki megoldású gépekhez, ezért nem várható hogy egyhamar egységesen programozhatnánk minden típuson! Így kénytelenek vagyunk mi, a felhasználók megtanulni a különböző nyelvi változatokat.

A helyzet azért nem teljesen kilátástalan, hiszen az alapvető utasításokat valamennyi BASIC ugyanolyan formában tartalmazza (PRINT, LIST, RUN stb.), és ha egy adott gépet már ismerünk, akkor könnyű átállni egy másik programozására is.

Az eltérések az egyes gépek különböző kiépítéséből is adódnak, mint pl. mágneslemez, sornyomtató, kazettás adattároló stb. Így nagyon fontos, hogy használat előtt alaposan tanulmányozzuk a gépkönyvet, és csak így, a lehetőségeket ismerve és figyelembe véve kezdjük el a munkát!

A BASIC programok kezelése

A BASIC programozási nyelv egyik jellemzője az *interaktív* programozási mód, amit párbeszédes üzemmódnak is neveznek. A programozó vagy felhasználó közvetlen kapcsolatban van a számítógéppel. A kapcsolatteremtés egyik nyelve a BASIC. A kapcsolatot biztosító hardver eszközök: a *billentyűzet* és a *képernyős megjelenítő*, ami legtöbb esetben egy közönséges TV.

A billentyűzeten begépelt karakterek, amikkel a számítógép működését szándékozunk befolyásolni, éppúgy megjelennek a képernyőn, mint a számítógép üzenetei, amik a felhasználó számára adnak információt. A *BASIC rendszer* ezt a folyamatot teremti meg. Ezzel segíti a programozást, és megkönnyíti a BASIC nyelv tanulását is.

A legelső lépés, ha a számítógépet használni szeretnénk, hogy megismerkedünk a számítógép *billentyűzetével*. A HT--1080Z típusú számítógép billentyűzetét és egyes billentyűk funkcióját az 1. mellékletben adjuk meg.

A billentyűzeten begépelt karaktersorozatot a BASIC rendszer csak akkor kezdi értelmezni, ha a *NEW LINE billentyűt* benyomjuk. Ezzel kell zámi a parancsokat és a programok utasítássorait is.

Nem megfelelő, a BASIC rendszer számára érthetetlen karaktersorozat begépelésekor a BASIC rendszer hibát jelez. A hibajelzésben szereplő betűkód a hiba jellegére utal. A *hibakódok* jelentését a 9. mellékletben adtuk meg.

A billentyűzetet ismerve, felhasználóként már használni is tudjuk a számítógépet, ha a kész programok kezelésére vonatkozó *BASIC parancsokat* megtanuljuk:

NEW – törli az előző BASIC programot.
CLOAD – a beépített magnetofonról beolvassa azt a programot, amelyik a mágnesszalagon következik.
RUN – elindítja a beolvasott vagy beírt programot.
LIST – kiírja a képernyőre a programot.
CSAVE "X" – X névvel ellátva a beépített magnetofon rögzíti a programot.

Kész programokat mágneskazettán tárolhatunk. A számítógépekhez beszerezhető programkazetták olyan programokat tartalmaznak, amelyek a legkülönbözőbb feladat megoldására teszik alkalmassá a számítógépet. A felhasználónak csak annyit kell tudni, hogy a *mágnesszalagon rögzített* programot hogyan kell *beolvasni* a számítógép központi memóriájába, és azt hogyan kell elindítani.

A BASIC rendszerünk a

```
READY
>_
```

kiírással jelzi, hogy parancsra vár.

A _ jel a *kurzor* pillanatnyi állását mutatja, azaz azt a karakterpozíciót, ahol megjelenik az a karakter, amit begépelünk.

Állítsuk a mágnesszalagot oda, ahol a BASIC program kezdõdik, és kapcsoljuk *lejátszásra* a magnetofont. (Az F1 nyomógomb legyen olyan állásban, hogy a magnetofon ne induljon el.)

Gépeljük be a CLOAD parancsot! Arról természetesen ne feledkezzünk meg, hogy a CLOAD szó begépelése után a NEW LINE billentyût kell lenyomnunk. A parancs végrehajtása csak akkor kezdõdik.

Helyes beolvasáskor a képernyõ jobb felsõ sarkában megjelenõ két csillag közül a jobb oldali villog. Miután a program betöltése sikeresen befejezõdik, a READY kiírás jelenik meg a képernyõn.

A programot a RUN paranccsal *indíthatjuk.*

A HT–1080Z számítógép kezelési parancsait a 3. mellékletben foglaltuk össze. Itt találunk bõvebb útmutatást a CLOAD parancs használatához is. A példákkal bemutatjuk a parancsok használatának lehetséges eseteit, amiket érdemes ki is próbálni.

A billentyûzet megismerésének legjobb módja, ha *kész programot gépelünk* be.

A tankönyv mintaprogramjai megfelelnek erre a célra, de találunk programokat a szakfolyóiratokban is. Akkor, ha még nem tudunk programozni, csak olyan programok begépelésével próbálkozzunk, amit kifejezetten az adott számítógépre írtak. Más géptípusra írt BASIC programok átírásához ismernünk kell mindkét számítógép BASIC nyelvét, tudnunk kell az eltérésekről.

A tankönyvben szereplõ mintaprogramokat a HT–1080Z számítógép BASIC nyelvén írtuk.

Mielõtt elkezdenénk a begépelést, a NEW paranccsal *töröljük* ki a memóriából az elõzõ BASIC programot.

A programok begépelésekor arra kell ügyelni, hogy a program utasítássorai mindig valamilyen pozitív egész számmal kezdõdnek. Ezek az *utasítássorok* sorszámai. Miután begépeltünk egy utasítássort, nyomjuk le a NEW LINE billentyût. Ezzel zárjuk a sort, amit a számítógép tárol.

Arról, hogy mit sikerült begépelnünk, úgy gyõzõdhetünk meg, hogy a LIST paranccsal *kiíratjuk* a programot.

A program indítása a RUN paranccsal történik.

Az elkerülhetetlen, hogy a program begépelésekor ne tévedjünk. Meg kell tanulnunk, hogy hogyan *javíthatjuk* a hibákat.

Mielõtt még lezárnánk a sort, a tévesen beütött karaktereket a ← billentyû lenyomásával törölhetjük.

A tárolt utasítássorokat

– törölhetjük,

– felülírhatjuk és

– az utasítássorok közé, eddig nem használt sorszámmal, utólag is beírhatunk utasítássorokat.

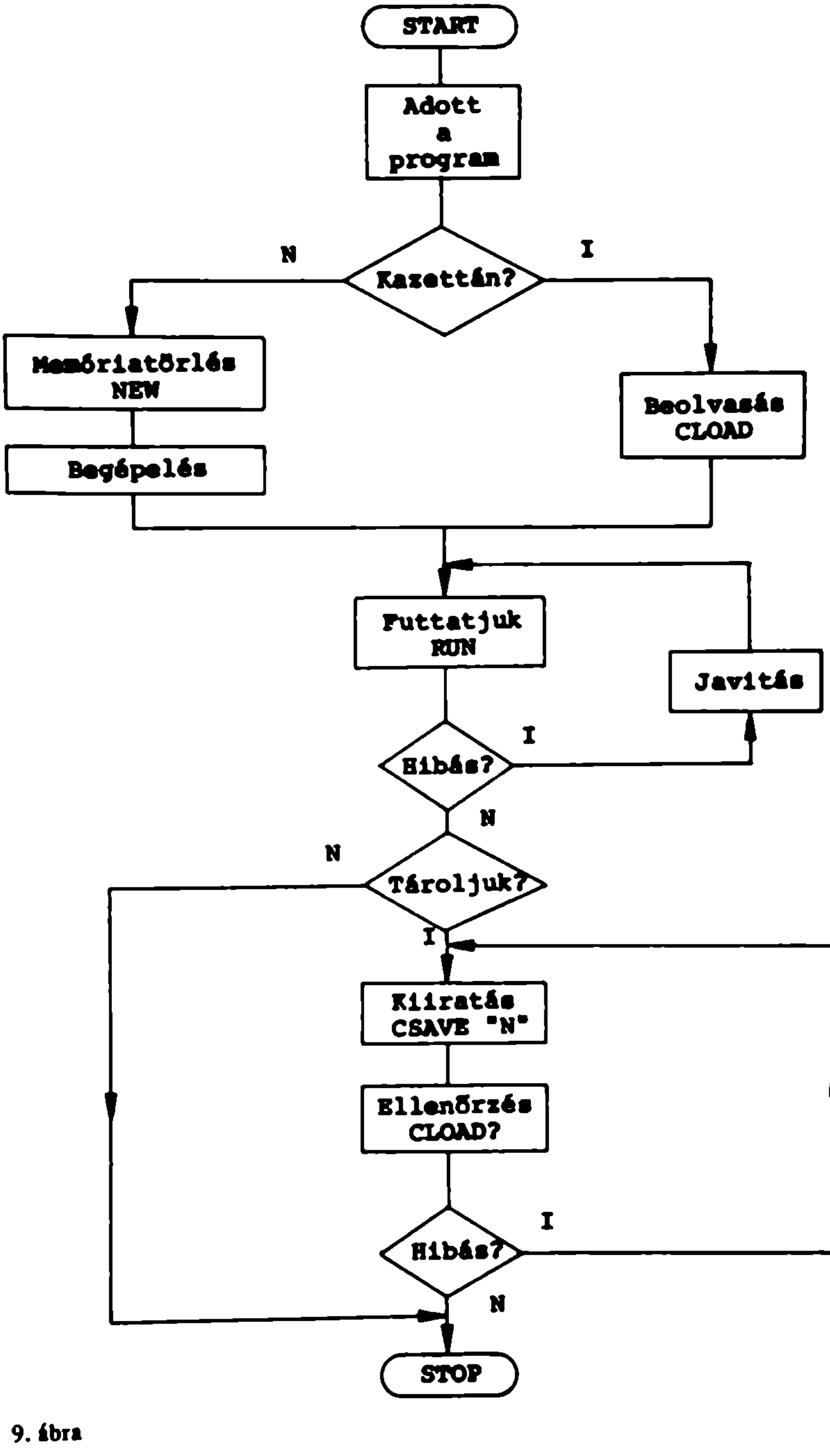
Egy programsort úgy *törölhetünk,* hogy begépeljük a törlendõ sor sorszámát.

A *felülírás* azt jelenti, hogy a hibás utasítássor helyett begépeljük a helyeset. Azonos sorszámmal begépelt utasítássorok közül a számítógép az utolsóként begépelt utasítássort õrzi meg.

A 3. mellékletben találunk olyan parancsokat, amik segíthetnek a hibajavításban: DELETE, EDIT:

A begépelt vagy a kazettáról beolvasott programot kazettán *tárolhatjuk.*

Állítsuk a szalagot oda, ahová a programot fel akarjuk venni, és kapcsoljuk a magnetofont felvételre.



9. ábra

A BASIC programok kezeléséhez tartozó műveleteket, és a műveletek végrehajtására szolgáló parancsokat a 9. ábra foglalja össze.

Az F1 nyomógombot úgy állítsuk be, hogy a magnetofon ne induljon el!

Adjuk ki a CSAVE "A" parancsot! A parancs hatására a felvétel elindul és a mágnesszalag rögzíti a programot. A program neve A lesz. Ez lenne a neve akkor is, ha az idézőjelek közé az ALMA szót írjuk, mert csak az első betűt veszi figyelembe a rendszer.

A felvételt úgy *ellenőrizhetjük*, hogy a szalagot visszacsévéljük a felvett program elejére, a magnetofont lejátszásra állítjuk, és kiadjuk a CLOAD? parancsot. Amennyiben a felvétel sikerült, az ellenőrzés közben a képernyő sarkában az egyik csillag villog, és a beolvasás végén kiíródik a READY szó.

Hiba esetén a felvételt megismételhetjük, mivel a program még a memóriában van.

Kérdések és feladatok

1. Melyik számítógép BASIC rendszerét ismerteti a tankönyv?
2. Hogyan jelzi a BASIC rendszer, hogy parancsra vár vagy programot gépelhetünk be?
3. Hogyan történik a mágnesszalagon tárolt program beolvasása?
4. Milyen paranccsal indíthatjuk el a BASIC programokat?
5. Hogyan rögzíthetjük mágnesszalagra a programot, és azt hogyan ellenőrizhetjük?
6. Melyik billentyűvel zárjuk a parancsokat és az utasítássorokat?
7. Ismertessük a program kezelésére vonatkozó parancsokat!
8. Mi a szerepe a NEW LINE és a SHIFT billentyűnek?
9. Hol találjuk a szóköz SPACE billentyűt?
10. Milyen javítási lehetőségeket ismerünk?
11. Gépeljük be a következő programot, amivel a billentyűzet használatát gyakorolhatjuk! A feladat az, hogy miután elindítottuk a programot, minél gyorsabban gépeljük be azt a karaktert, amelyik megjelenik a képernyőn!

```
10  REM BILLENTYUZET HASZNALATA
20  RANDOM
30  CLS:PRINT "HANY KARAKTERT GEPELSZ BE?"
40  PRINT "GEPELJ BE EGY EGESZ SZAMOT,
    ES"
50  INPUT "NYOMD LE A NEW LINE BILLENTYUT",N
60  FOR I=1 TO N
70  X=RND(59)+32
80  CLS
90  PRINT@480,CHR$(X)
100 PRINT@598,"GEPELD BE A KARAKTERT",
110 Y$=INKEY$:IF Y$=""THEN 110
120 CLS:L=L+1
130 IF Y$=CHR$(X)THEN PRINT@0,"ELTALALTAD"
    ELSE PRINT@0,"NEM SIKERULT
    GEPELD BE A KARAKTERT UJBOL!":GOTO90
140 FOR J=1 TO 300:NEXT J
```

```
150  NEXT I
160  CLS:PRINT@ 460,N,"KARAKTERT",
     L,"BEUTESSEL SIKERULT BEGEPELNED"
```

12. A 2. melléklet segítségével ismerkedjünk a BASIC nyelv alapelemeivel! Elemezzük a mintaprogramokat!

ADATBEOLVASÁS ÉS -KIÍRÁS

● Írjunk olyan programot, amivel kiírathatjuk a számítógéppel a nevünket, és azt, hogy hány évesek vagyunk!

Nagyon egyszerű a feladat, ha mi számoljuk ki az életkorát annak, aki a programot használni szeretné. Ekkor a programunk csak egy utasításból áll, amivel *kiíratjuk* a megfelelő szöveget.

```
10  PRINT"EMESE 16 EVES"
```

Általánosabb megoldásnak tekinthetjük, ha a felhasználó gépeli be azt, amit látni szeretne a képernyőn.

Kérjük a programmal a szöveget, és írassuk ki!

```
10  INPUT A$
20  PRINT A$
```

Az INPUT utasítás végrehajtásakor a képernyőn megjelenik egy kérdőjel, ami azt jelzi, hogy a program valamilyen *adatot vár.* Programunk kéri azt a szöveget, amit ki szeretnénk íratni a képernyőn. Miután begépeljük a karaktersorozatot és lenyomjuk a NEW LINE billentyűt, a számítógép tárolja azt az A$ (olvasd A dollár) változóban. Ezzel az INPUT utasítás végrehajtása befejeződött. Következik a PRINT utasítás, amivel *kiíratjuk* az A$ változó értékét, azaz azt a szöveget, amit begépeltünk. A karaktersorozatot *karakterláncnak* nevezzük.

Az A$ értéke a program minden futtatásánál más és más lehet, tehát változhat. Innen az elnevezés: *változó.*

Az előzőekből a következőket tanulhattuk meg:

– a program végrehajtása a sorszámok növekvő sorrendjében történik;

– INPUT utasítással adatokat kérhetünk;

– az A$ változó bármilyen karaktersorozatot jelenthet;

– PRINT utasítással kiírathatjuk az A$ változó értékét.

● Írjunk olyan programot, amelyikkel kiírhatjuk a nevünket és azt, hogy hány évesek vagyunk, ha megadjuk mikor születtünk!

Az előző megfogalmazásból az hiányzott, hogy nem adtuk meg, milyen *bemenő* (input) *adatokkal* dolgozzon a számítógép. A programban tehát kérni kell a születési évet, és ebből kell meghatároznunk az életkort.

```
 7  INPUT A$
12  INPUT X
23  PRINT A$,1984-X,"EVES"
```

Értelmezzük a programot!

7-es sor: Kéri a nevet, az A$ karakterlánc-változó értékét.

12-es sor: Kéri a születési évet, az X numerikus-változó értékét.

23-as sor: Kiírja a nevet, az életkort – miután azt kiszámította – és azt, hogy EVES

Mit kell észrevennünk?

Kétféle változót használunk a programban:

– A$ karakterlánc-változó értékének tetszőleges karaktersorozatot megadhatunk.
– X értéke csak numerikus adat lehet.

A PRINT utasítással kiírathatjuk:

– a változók értékét,
– az aritmetikai műveletek eredményét és
– tetszőleges karaktersorozatot.

A PRINT utasításban vesszővel vagy pontosvesszővel választhatjuk el a kiírandó elemeket.

Amikor a program az INPUT utasítással az adatokat kéri, csak a kérdőjel jelenik meg, és így a felhasználó nem tudhatja, hogy milyen adatot kér a program. Adjunk tájékoztatást a felhasználónak, és írassuk ki a programmal, hogy milyen adatot kell begépelni. Ezt többféleképpen oldhatjuk meg.

```
 3  PRINT "KEREM A NEVET"
 5  INPUT A$
12  INPUT "SZULETESI EVE",X
15  PRINT A$,1984-X,"EVES"
```

A 3-as sorban a PRINT utasítással íratjuk ki a tájékoztató szöveget, de elhelyezhetjük azt az INPUT utasításban is (12-es sor). Az INPUT utasításban a kiírandó szöveg után pontosvesszőt kell tenni, és ezután következhetnek azok a változók, amelyeknek az értékét kérjük.

```
10  REM ELETKOR
20  PRINT "NEV,SZULETESI EV"
30  INPUT N$,E
40  PRINT N$,1984-E,"EVES"
```

Egy INPUT utasítással több változó értékét is kérhetjük. A változókat vesszővel választjuk el egymástól (30-as sor).

Az első adat begépelése után, ha azt a NEW LINE billentyűvel zárjuk le, a rendszer két kérdőjel kiírásával jelzi, hogy még nem adtunk minden változónak értéket. Akkor, ha az INPUT utasításban több változó értékét kérjük, az adatokat úgy is megadhatjuk, hogy azokat vesszővel választjuk el, és csak az utolsó begépelése után nyomjuk le a NEW LINE billentyűt.

Vegyük észre, hogy az előző két program utolsó PRINT utasításaiban nem ugyanazokat az *elválasztójeleket* alkalmaztuk. A *kiírás formája* ennek megfelelően különböző lesz.

– *Pontosvessző* esetén a kiírás folyamatos.

– *Vessző* után a következő zóna elején folytatódik a kiírás. A képernyő 64 karakterpozíciója négy zónából áll, ezek kezdőpontjai: a 0., 16., 32. és a 48. karakterpozíció.

A program elején álló *REM utasítás* az értelmező program számára azt jelenti, hogy az utána következő karaktersorozatot ne vegye figyelembe. Mi most a program nevét írtuk ide.

A beolvasó utasítás:

INPUT változólista

jelentése: „Olvasd be a listában szereplő változók értékeit!"

A változólista elemeit vesszővel választjuk el egymástól.

INPUT "karakterlánc";változólista

jelentése: „Írd ki a karakterláncot és kérd a változólista elemeinek értékeit!"

A kiíró utasítás:

PRINT kiíratási-lista

jelentése: „Írd ki a kiíratási-lista lemeinek értékeit!"

A listaelem lehet:

– konstans: programban adott állandó.

A konstans lehet:

– numerikus: valamilyen szám
  1984 3 1415 6 E23
– karakterlánc: karaktersorozat, amit a PRINT utasításban idézőjel közé teszünk
  SZOVEG 1984. MARCIUS
– változó: adatok tárolására, azok azonosítására szolgáló szimbólum.

A változó lehet:

– numerikus: egy betű vagy egy betű és egy alfanumerikus karakter
  X A5 IK
– karakterlánc: a numerikus változótól a $ jel különbözteti meg
  A$ K3$ JP$
– aritmetikai kifejezés: konstansokon, változókon és függvényeken kijelölt műveletsor
  1984–E 2*R*3.1415
– TAB függvény: adott helyközzel léptetjük a kurzort.

Az utóbbi két listaelemről a következőkben lesz bővebben szó.

– a PRINT utasítás helyett elég a ?-et begépelni.

● Írjunk programot, amivel kiszámíthatjuk egy téglatest alakú tárgy felszínét, térfogatát és tömegét, ha adott az oldalélek hossza és a sűrűsége!

Az oldaléleket jelöljük a -val, b-vel és c-vel, a sűrűséget d-vel. Ekkor a test felszíne:

$$F = 2ab+2ac+2bc,$$

térfogata:

$$V = abc,$$

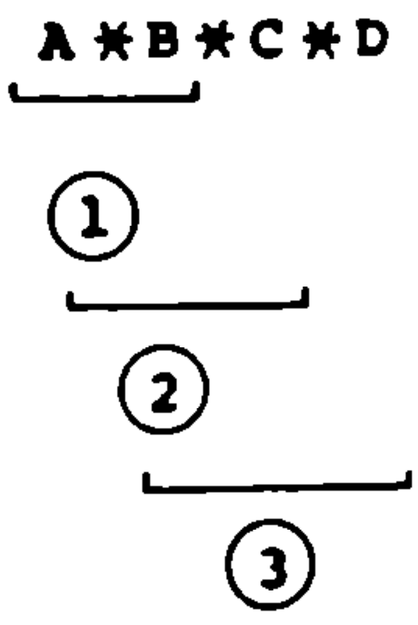
tömege:

$$m = abcd.$$

```
10  REM TEGLATEST
20  INPUT "OLDALELEK",A,B,C
30  INPUT "SURUSEG",D
40  PRINT "FELSZIN=",2*A*B+2*A*C+2*B*C
50  PRINT "TERFOGAT=",A*B*C
60  PRINT "TOMEG",A*B*C*D
```

A programból is látszik, hogy az aritmetikai kifejezések hasonlóképpen írandók a programban, mint ahogyan a matematikában a paraméteres kifejezéseket leírjuk. A műveletek végrehajtása, a kifejezés értelmének kiszámítása azonos a matematikai *sorrendiségi szabállyal.*

A programban szereplő két kifejezés értékének kiszámítását mutatja a 10. ábra, ahol a számokkal a műveletek sorrendjét jelöltük. Először a szorzásokat, majd az összeadásokat végzi el a számítógép, balról jobbra haladva.



10. ábra

Az osztás (jele: /) a szorzással, a kivonás az összeadással azonos sorrendiségű művelet. Legelőször a hatványozást végzi el a számítógép (jele: ↑).

A műveletek sorrendje tehát a következő:

↑ hatványozás
* szorzás
/ osztás
\+ összeadás
– kivonás

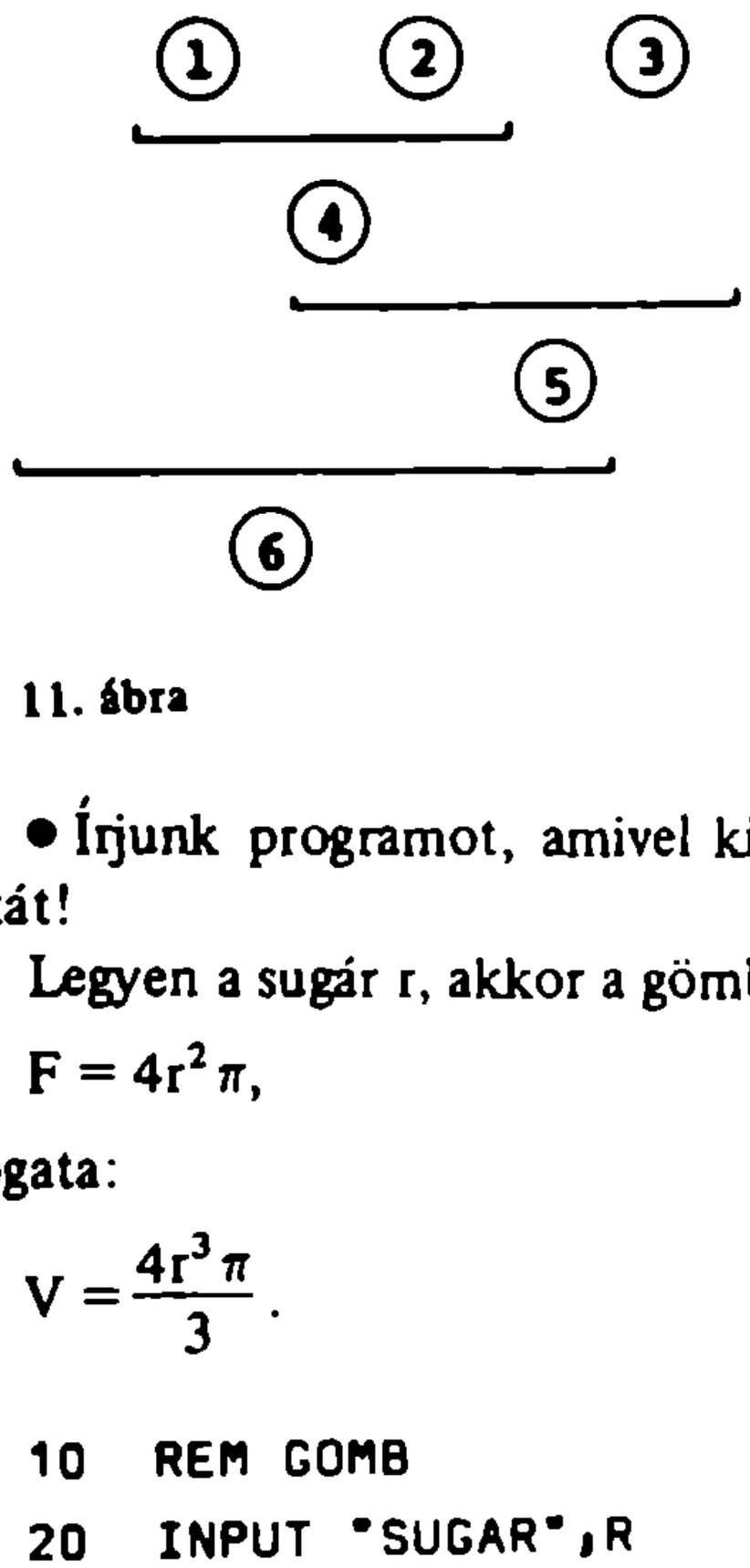
A műveletek végrehajtásának sorrendjét zárójelek alkalmazásával tudjuk megváltoztatni.

A test felszínét kiszámíthatjuk a

```
40  PRINT "FELSZINE",2*(A*B+A*C+B*C)
```

utasítással is.

Ekkor a végrehajtás sorrendje a 11. ábrán bemutatott:



11. ábra

● Írjunk programot, amivel kiszámíthatjuk egy adott sugarú gömb felszínét és térfogatát!

Legyen a sugár r, akkor a gömb felszíne:

$F = 4r^2\pi$,

térfogata:

$$V = \frac{4r^3\pi}{3}.$$

```
10  REM GOMB
20  INPUT "SUGAR",R
30  PRINT "FELSZINE",4*R↑2*3.1415
40  PRINT "TERFOGATA",4*R↑3*3.1415/3
```

A kifejezésekben háromféle számtípust használhatunk, és ilyen alakban adhatjuk meg a numerikus adatokat is:

– egész: 4 3 1984

– tizedespontos: 3.1415

Az előző számok

– lebegőpontos alakja: 0.4E1 30E–1 19.84E2 3141E–4

Ezt az alakot általában nagyon nagy, vagy nagyon kis abszolútértékű számok esetén használjuk.

$6\cdot10^{23}$	6E23
$6{,}67\cdot10^{-11}$	6.67E–11

Az E betű a 10 kitevőjét választja el az értékes jegyektől.

```
10  REM HATVANYOZAS
20  INPUT "ALAP",A
30  INPUT "KITEVO",K
40  PRINT "HATVANY ERTEKE",A ↑ K
```

• Írjunk programot, amivel kitapasztalhatjuk, hogy milyen értéke lehet a hatványalapnak, és milyen a kitevőnek!

A program segítségével meggyőződhetünk arról, hogy a számítógép pozitív hatványalap esetén a 0, negatív és törtkitevőjű hatványozást is végrehajtja. Amennyiben a kitevő egész, negatív alap esetén is kiszámítja a hatvány értékét.

• Számítsuk ki a 14,36 cm sugarú kör kerületét és területét!

Ilyen egyszeri felhasználásra programot írni felesleges. A PRINT utasítást parancsként használva a következőképpen oldhatjuk meg a feladatot:

```
PRINT 2 * 14.36 * 3.1415, 14.36 * 2 * 3.1415
```

Sorszám nélkül begépelt utasítást, a *parancsot* a számítógép azonnal végrehajtja. Ezt a használati módot *kalkulátor üzemmódnak* is nevezik. Számítások gyors elvégzésére és az utasítások hatásának a vizsgálatára használjuk.

Kérdések és feladatok

1. Az 1. mellékletből keressük ki azokat az alapelemeket, amikről szó volt ebben a fejezetben!

2. Miért nem egyesével növeljük a programutasítás sorainak sorszámait?

3. Értelmezzük a következő programot, és egészítsük ki magyarázó szövegek kiíratásával!

```
1   INPUT R
2   PRINT 2*R*3.1415
3   PRINT R↑2*3.1415
```

4. Mennyi lehet a sorszám maximális értéke?

5. Hogyan jelölhetjük a numerikus-változókat, és hogyan a karakterlánc-változókat?

6. Miben különbözik a numerikus-változó a karakterlánc-változótól?

7. Adjuk meg különböző alakban a következő numerikus adatokat!

14,37 $44{,}3 \cdot 10^{-4}$ −0,0001 30000 248,7 123456789

Megoldásuk helyességét a következő programmal ellenőrizhetük:

```
10  INPUT X
20  PRINT X
```

8. Írjunk programot

a) az egyenes körhenger felszínének és térfogatának a kiszámítására, ha adott az alapkör sugara és a henger magassága:

b) az $ax+b = c$ alakú egyenlet megoldására;

c) a gömb sugarának a kiszámítására, ha ismerjük a gömb felszínét!

9. Hogyan jelzi a rendszer, hogy hibás adatot adtunk meg?
10. Hogyan javíthatjuk a hibásan begépelt adatot?
11. Mi lesz a PRINT -3↑2 parancs eredménye?

ÉRTÉKADÁS

• Írjunk programot, amivel meghatározhatjuk az egyenes körhenger alakú test felszínét, térfogatát és sűrűségét, ha adott az alapkör sugara, a magasság és a test tömege! A körhenger:

felszíne: $2r^2\pi+2r\pi h = 2\pi(r+h)$

térfogata: $r^2\pi h$, ha

a henger alapkörének sugara r, magassága h. Jelöljük a test térfogatát V-vel, a test tömegét m-mel. Ekkor a

sűrűség: $\frac{m}{V}$.

Oldjuk meg a feladatot eddigi ismereteink alapján!

```
10  REM KORHENGER-1
20  INPUT "ALAPKOR SUGARA,MAGASSAG",R,H
30  INPUT "TOMEG",M
40  PRINT "FELSZIN",2*R*3.1415*(R+H)
50  PRINT "TERFOGAT",H*R↑2*  3.1415
60  PRINT "SURUSEG", M/(H*R↑2*3.1415)
```

Az 50-es sorban a PRINT utasítás kiszámítja és kiírja a test térfogatának mérőszámát. Mivel ezt az értéket nem tárolta a számítógép, ezért a sűrűség meghatározását ismét ki kell számítania, nekünk pedig még egyszer le kellett írnunk a test térfogatát kiszámító kifejezést, amikor a programot írtuk.

Járjunk el hasonlóan, mint ahogyan azt a feladat matematikai leírásában tettük. Számíttassuk ki a térfogat értékét, és helyezzük el azt a V változóban! Ezt oldja meg a LET utasítás, amit *értékadó utasításnak* neveznek.

A π értékére is többször szükség van. Adjuk a P változónak a π értékét, és akkor csak a P változóra kell hivatkoznunk a kifejezésekben.

```
10  REM KORHENGER-2
20  INPUT "ALAPKOR SUGARA,MAGASSAG",R,H
30  INPUT "TOMEG",M
40  LET P=3.14159
50  PRINT "FELSZIN",2*R*P*(R+H)
60  LET V=H*R↑2*P
70  PRINT "TERFOGATA",V
80  PRINT "SURUSEG",M/V
```

Az értékadó utasítás

LET változó = aritmetikai kifejezés

jelentése: „Legyen a változó értéke a jobb oldalon álló aritmetikai kifejezés kiszámított értékével egyenlő!"
vagy másképpen fogalmazva:
„Vegye fel az egyenlőségjel bal oldalán álló változó a jobb oldali kifejezés kiszámított értékét!"

A LET utasításszó elhagyható.

A változók értékadása többféleképpen történhet. Az *INPUT utasítással* a program futtatásakor *kérjük* a bemenő adatokat, akkor adunk értékeket a változóknak, a *LET utasítással* a programban *adjuk* meg a változók értékeit. Ha egy változóra úgy hivatkozunk, hogy korábban nem kapott értéket – amit úgy is mondunk, hogy nem definiáltuk a változót –, akkor a változó értéke 0 lesz.

• Értelmezzük a következő programot!

```
10  REM ERTEKADAS
20  PRINT I
30  INPUT I
40  PRINT I
50  LET I=I+1
60  PRINT I
```

Futtatva a programot, három számot ír ki: 0-t, azt amit I-nek adtunk az INPUT utasítással és ennél 1-gyel nagyobbat.

A 20-as sorban az I értékre úgy hivatkoztunk, hogy az I-nek még nem adtunk értéket. Ezért írja ki a 0-t. Az 50-es sorban az I értékét 1-gyel megnöveltük. Ez az utasítás jól mutatja, hogy az *egyenlőségjel* a LET utasításban nem egyenlőséget jelöl, hanem *az értékadás jele*. Adjuk meg szóban ezt az utasítást: „Legyen I értéke I+1-gyel egyenlő!" A végrehajtás során az történik, hogy az I+1 kifejezést kiszámítja a számítógép, és azt az I változóban elhelyezi. Ezzel I értéke módosul.

• Írassuk ki a hosszúság alapmértékegységének tized-, század- és ezredrészének megnevezését!

```
10  REM MERTEKEGYSEGEK
20  A$="METER"
30  K$="CENTI"
40  L$="MILLI"
50  X$="DECI"+A$
60  Y$=K$+A$
70  L$=L$+A$
80  PRINT X$,Y$,L$
```

A programban karakterlánc-változóknak adunk értéket a LET utasítással. A LET szót már nem írtuk ki. A + jellel *a karakterláncok összefűzésének műveletét* jelöljük. Két karakterláncból előállítunk egy újabbat, amit egy karakterlánc-változóban helyezünk el. Vegyük észre, hogy az L$ változó értéke az L$ = L$+A$ utasítás hatására módosul.

• Határozzuk meg egy adott sugarú kör kerületét és területét a lehető legnagyobb pontossággal!

```
10  REM PONTOSSAG
20  INPUT "SUGAR",R
30  P=3.14159
40  KE=2*R*P
50  TE=R↑2*P
60  PRINT "KERULET",KE,"TERULET",TE
```

A π értékét hiába adjuk meg pontosabban, csak 7 értékes számjegyet tudunk a P változóba elhelyezni, és ennyi értékes számjegye lesz a KE és a TE változók értékeinek is.

Írjuk a programba az

```
15  DEFDBL R,P,K,T
```

utasítást. Az utasításban felsorolt betűkkel kezdődő változókat az utasítás hatására *kétszeres pontosságú változóként* kezeli a rendszer. A kétszeres pontosságú változók értékes számjegyeinek száma maximálisan 16. A DEFDBL utasítás helyett, a kétszeres pontosságú változók megjelölésére használhatjuk a # (számjel) karaktert is. Ekkor a programban az R#, P#, KE# és TE# változókat használjuk.

Megkülönböztetésül az eddig használt változókat *egyszeres pontosságú* változóknak is nevezzük. Az egyszeres pontosságú változók kijelölése DEFSNG utasítással vagy a !-lel történik: A!, X2!, MN!. Alaphelyzetben a változók egyszeres pontosságúak.

● Írjunk programot egy 8 bites bináris szám (bájt) tízes számrendszerbeli értékének kiszámítására!

Jelöljük a bináris szám számjegyeit az ABC első nyolc betűjével! A szám decimális értékét az

$$(1)\ a\cdot 2^7+b\cdot 2^6+c\cdot 2^5+d\cdot 2^4+e\cdot 2^3+f\cdot 2^2+g\cdot 2+h$$

kifejezés vagy más formában felírva a

$$(2)\ ((((((a\cdot 2+b)\cdot 2+c)\cdot 2+d)\cdot 2+e)\cdot 2+f)\cdot 2+g)\cdot 2+h$$

kifejezés értéke adja. Ez utóbbi formát Horner elrendezésnek nevezik.

```
10  REM BYTE DECIMALIS ERTEKE
20  CLS:INPUT "8 BITES BINARIS SZAM
    SZAMJEGYEI",A,B,C,D,E,F,G,H
30  X=A*2↑7+B*2↑6+C*2↑5+D*2↑4+E*2↑3+
    F*2↑2+G*2+H
40  PRINT "A DECIMALIS ERTEKE",X
50  X=((((((A*2+B)*2+C)*2+D)*2+E)*2+F)*2+G)
    *2+H
60  PRINT "A DECIMALIS ERTEKE",X
70  Z=A:Z=Z*2+B:Z=Z*2+C:Z=Z*2+D
80  Z=Z*2+E:Z=Z*2+F:Z=Z*2+G:Z=Z*2+H
90  PRINT "A DECIMALIS ERTEK",Z
```

A programban a két összetett kifejezés értékének a kiszámításán kívül megoldjuk a feladatot úgy is, hogy a számításokat *azonos alakú* utasítások sorozatára bontjuk. A 70-es és a 80-as utasítássorban szereplő utasítások hatására az történik. hogy a Z változó érté-

kének kétszereséhez hozzáadjuk a következő számjegyet, és így megkapjuk a Z újabb értékét.

Ezt a megoldást a (2) kifejezés kiszámítása sugallja.

A programban a változók értéke csak egész szám lehet, ezért használhatunk *egész változókat* is. Az egész változókat a utasítással vagy a változók neve után írt %-jellel jelölhetjük ki: A%, X%. Az egész változók −32768 és 32767 közötti értékeket vehetnek fel.

A változók típusától függ, hogy hány memóriarekeszt használ fel a rendszer a változók értékének tárolására. Egész változó esetén 6-ot.

Egyszeres pontosságú változó értékéhez 7, kétszeres pontosságú változó tárolásához 11 rekesz szükséges. Egy rekeszbe 8 bit írható, azaz 1 bájt.

● Írjunk programot két pozitív egész szám maradékos osztásának a végrehajtására!

Tudnunk kell, hogy az egész változók a tört értékének csak az egész részét tárolják. A két adott szám hányadosának egész része megadja a maradékos osztás hányadosát. A hányados ismeretében kiszámíthatjuk a maradékot is.

```
10  REM MARADEKOS OSZTAS
20  INPUT "A KET SZAM;A > B",A,B
30  DEFINT H
40  H=A/B
50  M=A-B *H
60  PRINT "HANYADOS",H
70  PRINT "MARADEK",M
```

Kérdések és feladatok

1. Milyen utasításokkal adhatunk értékeket a változóknak?
2. Milyen típusú változókat ismerünk?
3. Hány karakterből állhat egy karakterlánc? Próbáld ki!
4. Hasonlítsuk össze az INPUT és a LET utasítást!
5. Mi lesz az értéke a definiálatlan karakterlánc-változónak?
6. Hogyan történhet az egész, az egyszeres pontosságú, a kétszeres pontosságú és a karakterlánctípusú változók kijelölése?
7. Mennyi lehet az egyszeres pontosságú változók értékes jegyeinek maximális száma?
8. Mi határozza meg az eredmény pontosságát?
9. Készítsünk programot az ax+b= cx+d alakú egyenlet megoldására!
10. Írjunk programot egy gondolt szám kitalálására!

A játék legyen a következő:

Gondolj egy számot!
Adj hozzá 3-at!
Szorozd meg 2-vel!
Végy el belőle 5-öt!
Gépeld be a kapott eredményt!

A gép „találja ki”, hogy mire gondoltunk! Használhatjuk az A%, A és A# változókat ugyanabban a programban?

11. Állítsuk elő program segítségével a tizedesszámok törtrészét!

12. Írjunk programot, ami, ha megadjuk az időtartamot órában, percben, másodpercben, akkor kiírja másodpercben!

13. Az egész számok hatványozásakor az eredmény nem mindig egész. Hogyan oldhatjuk meg ezt a problémát?

A PROGRAM VEZÉRLÉSE

Eddigi mintaprogramjaink ún. *soros programok* voltak, amelyekben az utasítások végrehajtásának sorrendjét a sorszámok egyértelműen meghatározták. A soros programok végrehajtása a legkisebb sorszámú utasítással kezdődik, és a sorszámok növekvő sorrendjében folytatódik.

Akkor, ha ezen a sorrenden változtatni akarunk, ha azt szeretnénk elérni, hogy ne a sorszám szerint következő utasítás végrehajtására kerüljön sor, hanem attól nagyobb vagy kisebb sorszámú utasítás következzen, akkor *vezérlő utasításokat* kell használnunk. Ezekkel az utasításokkal befolyásolhatjuk az utasítások végrehajtásának sorrendjét, azaz „vezérelhetjük" a programot.

Feltétel nélküli vezérlés

A feltétel nélküli vezérlés utasításszava a GOTO. Magyar jelentése: menj, ugorj.

Az utasításszó után írt sorszámmal azt adjuk meg, hogy hová „ugorjon", azaz hogy melyik utasítássor végrehajtása következzen. A következő mintaprogramban a GOTO 10 utasítással a program végéről ugrunk a 10-es sorra, a program elejére, és így nem kell mindig újraindítani a programot.

Feltétel nélküli vezérlő utasítás:

GOTO sorszám

jelentése: „Az adott sorszámú utasítássor végrehajtásával folytatódjon a program!"

● Írjunk programot, amivel kiszámíthatjuk a négyzetes egyenes csonkagúla felszínét és térfogatát, az alap és a fedőnégyzet élhosszából (a és b) és a csonkagúla magasságából (m)!

A csonkagúla térfogata:

$$V = \frac{m}{3}\,(a^2+ab+b^2),$$

felszíne:

$$F = a^2+b^2+(a+b)\sqrt{4m^2+(a-b)^2}\ .$$

```
10  INPUT A,B,M
20  V=M*(A↑2+A*B+B↑2)/3
30  F=A↑2+B↑2+(A+B)*SQR(4*M↑2+(A-B)↑2)
40  PRINT V,F
50  GOTO 10
```

A GOTO 10 utasítással a programunkat „végtelenítettük", azaz egy végtelen *ciklust*, ismétlődést hoztunk létre. Ez a program csak akkor áll le, ha lenyomjuk a BREAK billentyűt. Ezt nevezzük *külső programmegszakításnak.*

• Írassuk ki programmal a páratlan természetes számokat!

A megoldás lényege, hogy egy változó értékét kell mindig növelni 2-vel, és azt kell kiíratni. Ezeknek az utasításoknak az *ismételt végrehajtását* a GOTO utasítással érjük el.

Az első szám, amit ki kell írnia a programnak az 1. Ez lesz a *változó kezdőértéke*, és erről nekünk kell gondoskodnunk egy értékadó utasítással.

```
10  I=1
20  PRINT I
30  I=I+2
40  GOTO 20
```

Vegyük észre, hogy most nem ismétlődik mindegyik utasítás a végrehajtás során, mint az előző programban. Az ismételten végrehajtott utasítások (20-as és 30-as sorok) alkotják a *ciklusmagot*. A GOTO utasítás végrehajtása is ismétlődik, de ezt nem tekintjük a ciklusmag részének. Ez az utasítás az ismétlődést biztosítja.

A következő részben arról is szó lesz, hogy milyen utasítást kell beírnunk a programba, ha csak egy adott számig akarjuk kiíratni a páros számokat, tehát véges számszorzattal akarjuk végrehajtani a ciklusmagot.

Feltételes vezérlő utasítás

A matematikai *relációjeleket* a BASIC programozási nyelv is használja. Jelentésük azonos, csak a leírásuk különböző. Alkalmazkodni kell a gépelés kötöttségéhez. Összetett relációjeleket két karakterrel adunk meg.

= egyenlő
< kisebb
> nagyobb
<= kisebb vagy egyenlő
>= nagyobb vagy egyenlő
<> nem egyenlő.

A relációjelekkel konstansok, változók és aritmetikai kifejezések értékeit hasonlítjuk össze. Azok viszonyáról állítunk valamit. Ilyen *relációkkal* kifejezett állítások:

A > B A nagyobb B-nél

I <= 100 I kisebb, mint 100 vagy egyenlő 100-zal.

B↑2-4*A*C > 0 a relációjel bal oldalán álló kifejezés értéke nagyobb mint 0.

Miután a változóknak minden alkalommal más-más értéket adhatunk, és változhat értékük a program futása közben is, ezért ezek az állítások vagy *igazak*, vagy *hamisak*. A relációk logikai értéke, a relációkban szereplő kifejezések értékétől függ.

A BASIC rendszer a relációk logikai értékeit numerikus értékekkel azonosítja: –1 az IGAZ, 0 a HAMIS állításnak felel meg.

Próbáljuk ki ezt a következő programmal!

```
10  INPUT A,B
20  PRINT A<B
```

A logikai értékektől függő vezérlésátadó utasítást *feltételes vezérlésátadó utasításnak* nevezzük.

Ezzel az utasítással fogalmazhatjuk meg a számítógép számára azt, hogy egy feltétel, egy állítás a logikai értékétől függően mit csináljon, milyen utasításokat hajtson végre.

• Írassuk ki 100-ig a páratlan természetes számokat!

Most oldjuk meg azt a problémát, hogy a páratlan számok kiírása csak egy adott határig történjen. Addig, ameddig az $I < 100$ egyenlőtlenség teljesül. Ezt a *feltételünket* az IF–THEN utasítással adjuk meg.

```
10   I=1
20   PRINT I
30   I=I+2
40   IF I<100 THEN 20
50   PRINT "VEGE"
```

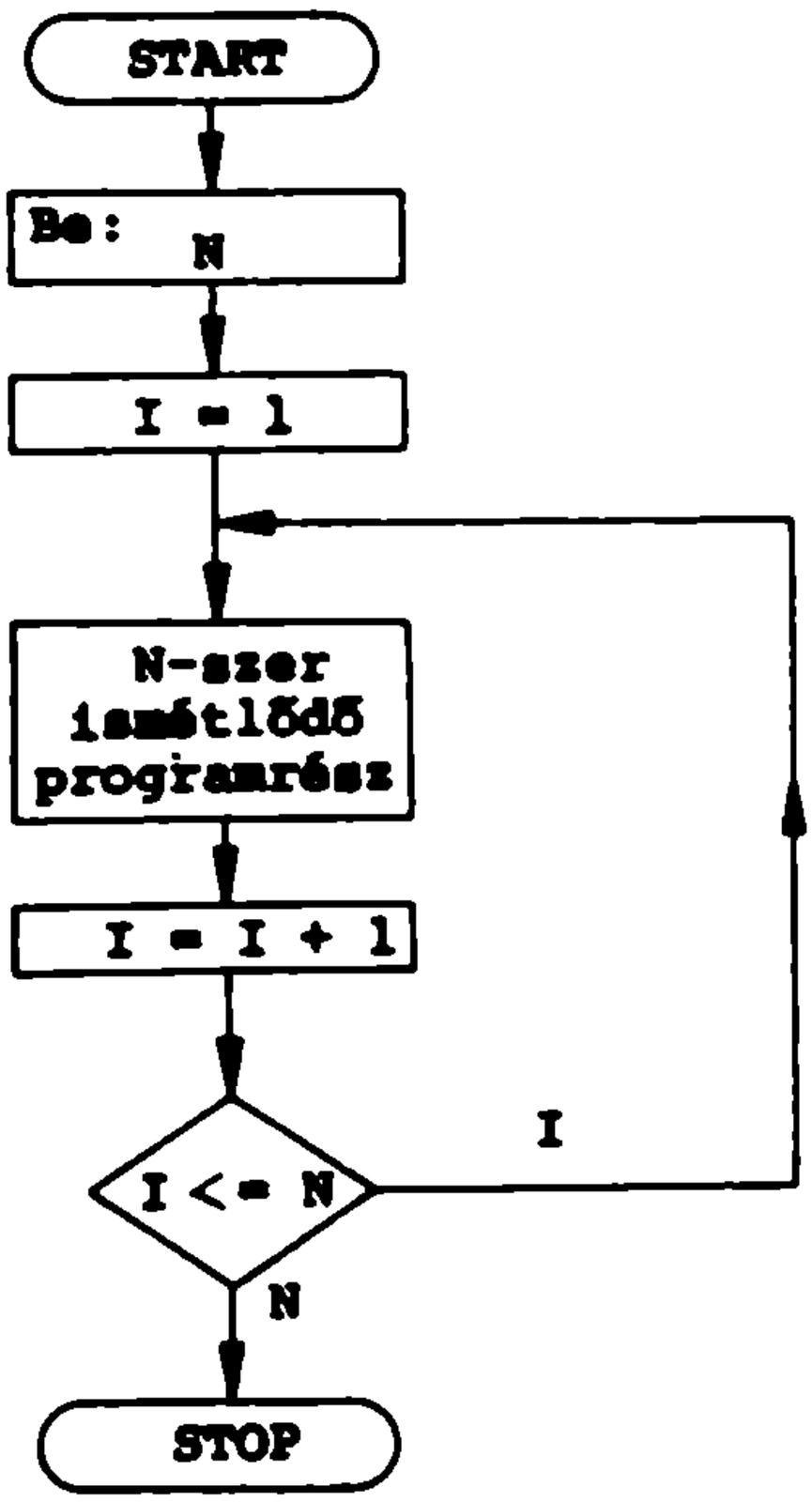
Az IF és a THEN utasításszavak magyar jelentését ismerve, az utasítást így olvashatjuk: „HA I kisebb mint 100, AKKOR ugorj a 20-as utasítássorra!"

Ez azt jelenti, hogy amennyiben a reláció logikai értéke IGAZ, akkor az I értékét írja ki a program, ha HAMIS, akkor az 50-es utasítássort hajtja végre a számítógép és kiírja a VEGE szót.

• Írassuk ki N-szer a BASIC szót!

Az előző program alapján könnyen megoldható a feladat, ha átfogalmazzuk:

Írassuk ki 1-től N-ig a természetes számokat! Ez azt jelenti, hogy N darab számot kell kiíratnunk, azaz N-szer kell a ciklusmagot végrehajtania a programnak. A ciklusmagban elhelyezzük a PRINT „BASIC" utasítást és ezzel megoldottuk a feladatot (12. ábra).



12. ábra

```
10   INPUT N
20   I=1
```

```
30  PRINT I,"BASIC"
40  I=I+1
50  IF I<=N THEN 30
```

Az előző két mintaprogramban *ciklust szerveztünk*, azaz előállítottunk egy olyan programrészletet, amellyel utasítások ismételt végrehajtását írtuk elő. Az IF–THEN utasítás hatására vagy *visszaugrunk* egy kisebb sorszámú utasítássorra, és így újból végrehajtja a számítógép ettől kezdődően az utasításokat, vagy folytatódik a program az IF–THEN utasítást követő utasítással. Az utóbbi esetben azt mondjuk, hogy *kiléptünk* a ciklusból. A IF–THEN utasítással tehát megoldható, hogy valamilyen feltételtől függően különböző utasításokat hajtson végre a számítógép. Az IF–THEN utasítás *a feltételt megvizsgálja* és ennek megfelelően történik a döntés.

A következő programokban a vizsgálaton lesz a hangsúly. A vizsgálat eredményétől függően *elágaztatjuk* a programot, és így különböző utasításssorozatok végrehajtását írhatjuk elő.

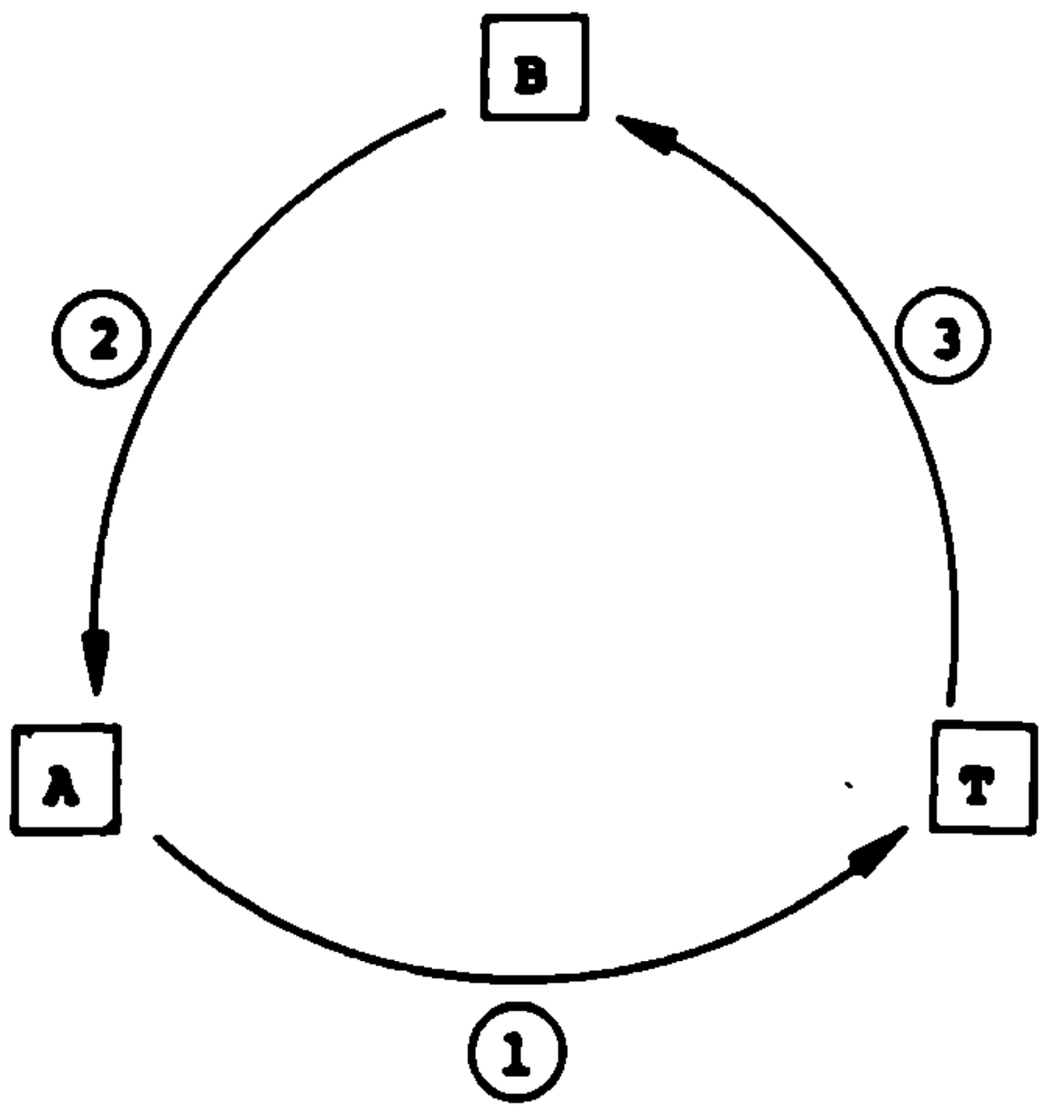
● Készítsünk programot, amivel két tetszőleges számot növekvő sorrendben íratunk ki!

```
10  INPUT A,B
20  IF A>B THEN T=A:A=B:B=T
30  PRINT A,B
```

Akkor, ha az A változónak nagyobb értéket adunk, kicseréljük a két változó értékét, és így a kiíráskor mindig az A változó értéke lesz kisebb.

Az IF–THEN utasításban a sorszám helyett bármilyen utasítássorozatot írhatunk a *THEN szó után*. Amennyiben a reláció *logikai értéke IGAZ*, ezeket az utasításokat végrehajtja a számítógép. Amikor sorszámot írunk a THEN után, akkor az is egy utasítás, csak nem írjuk ki a GOTO szót.

A két változó értékének a cseréjéhez felhasználtunk egy munkaváltozót, amiben megőriztük az egyik értéket (13. ábra).



13. ábra

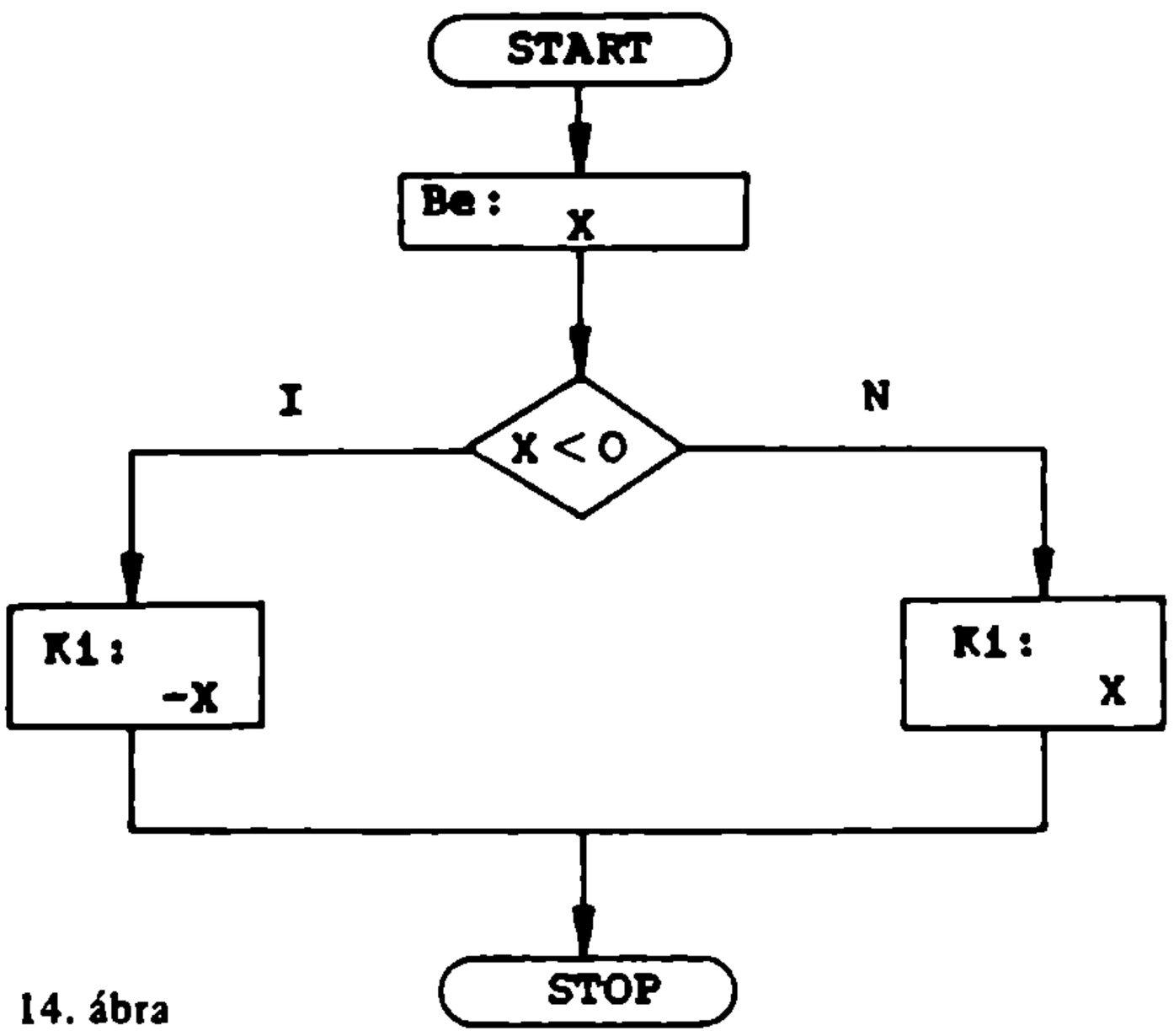
● Írassuk ki programmal egy szám abszolút értékét!

```
10  INPUT  X
20  IF  X < Ø THEN PRINT - X ELSE PRINT X
```

A 20-as sorban a következő utasítást adtuk: „HA X értéke kisebb, mint nulla, AKKOR írjuk ki a szám negatívját, EGYÉBKÉNT magát a számot!"

Az *ELSE utasításszó után* azokat az utasításokat írhatjuk, amiket a *reláció HAMIS* értékénél kell végrehajtania a programnak.

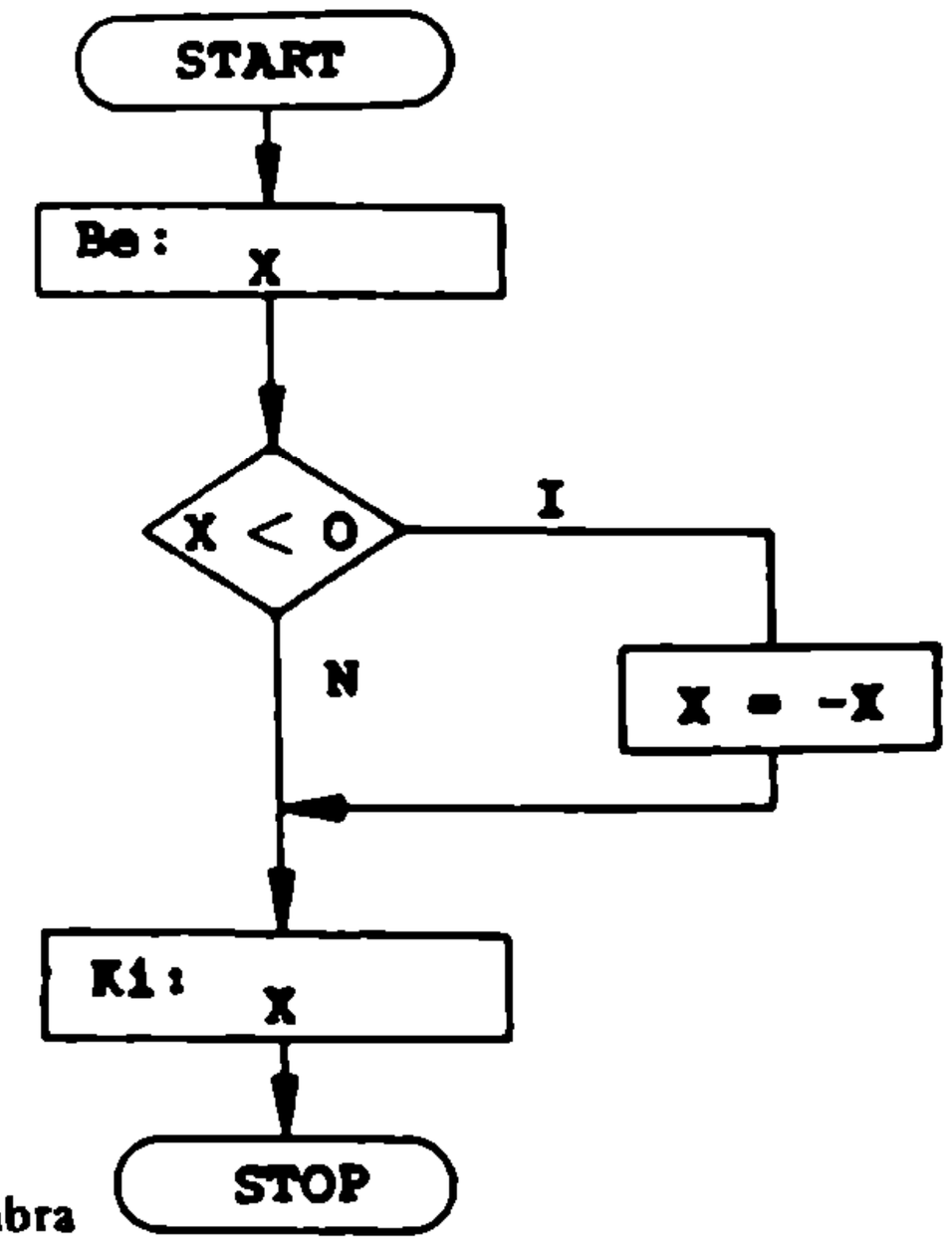
Az ELSE használatát nem mindegyik BASIC rendszer engedélyezi. Használata akkor előnyös, amikor két különböző műveletsort kell végrehajtanunk a feltételtől függően. A folyamatábra ebben az esetben a következő (14. ábra):



14. ábra

A feladat megoldható az ELSE használata nélkül is.
A program folyamatábrája a következő (15. ábra):

```
10  INPUT X
20  IF X<Ø THEN X=-X
30  PRINT X
```



15. ábra

A feltételes vezérlésátadó utasítás

IF állitás THEN utasítások ELSE utasítások

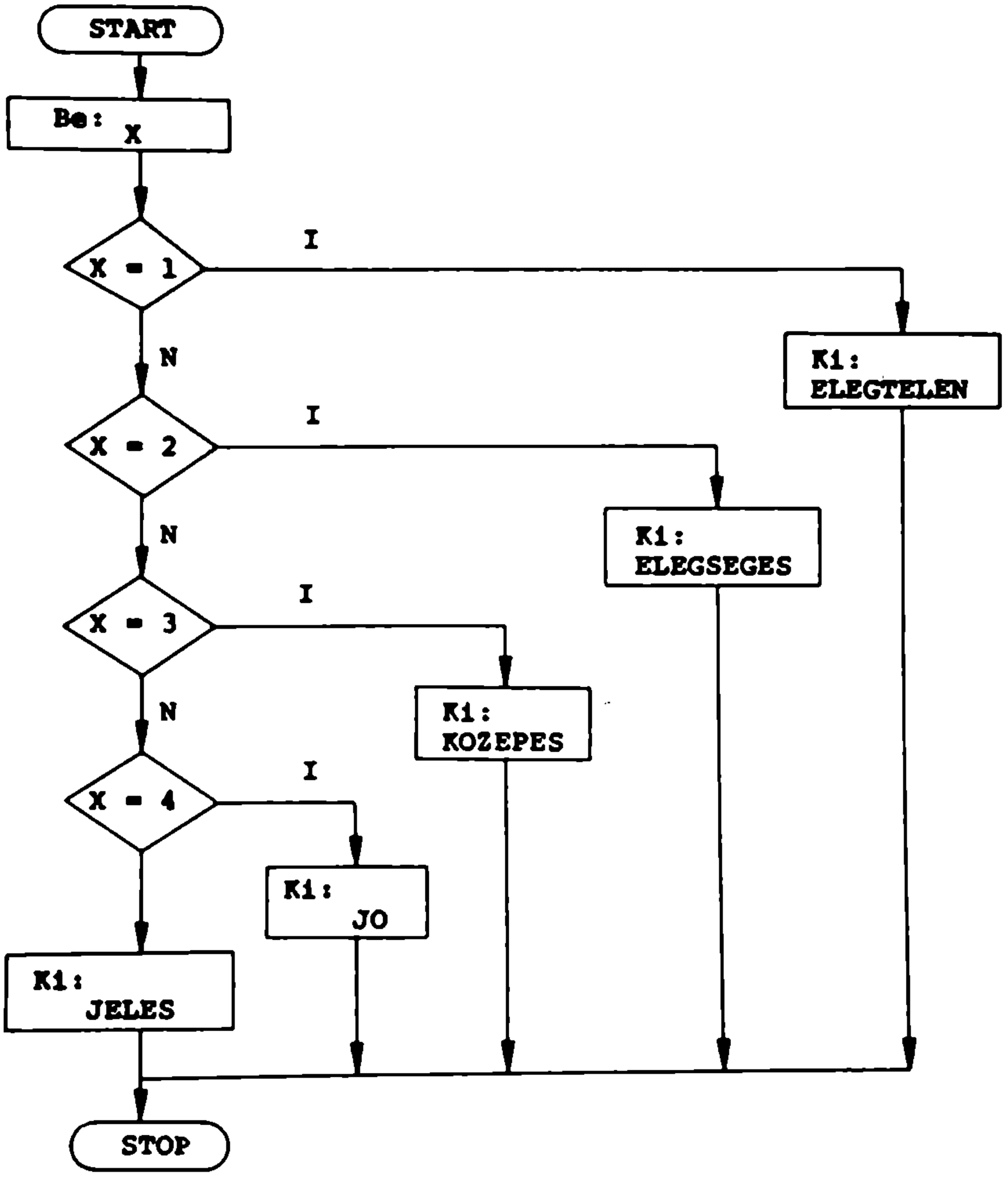
jelentése: „Ha az állítás igaz, akkor a THEN szó utáni utasításokat, ha hamis, akkor az ELSE szó utáni utasításokat kell végrehajtani!"

– az ELSE szót nem kötelező használni.

Amennyiben csak a GOTO utasítás szerepelne a THEN vagy az ELSE szavak után, elég csak a sorszámot megadni, a GOTO szó elhagyható.

• Írjunk programot, amivel kiírhatjuk szavakkal az osztályzatot, ha azokat számokkal adjuk meg!

A számmal adott osztályzatok vizsgálata, és a szöveges kiírás a következő algoritmus szerint történik (16. ábra):



16. ábra

```
10   REM OSZTALYZATOK-1
20   INPUT "OSZTALYZAT";X
30   IF X=1 THEN PRINT "ELEGTELEN":GOTO 70
```

```
40   IF X=2 THEN PRINT "ELEGSEGES";STOP
50   IF X=3 THEN PRINT "KOZEPES";END
60   IF X=4 THEN PRINT "JO" ELSE
     PRINT "JELES"
70   REM EZ LESZ AZ UTOLSO SOR
```

A *program befejezését* többféleképpen oldottuk meg. A GOTO 70 utasítással a program utolsó sorára ugrunk és így fejeződik be a program. Az utolsó sort csak azért írtuk a mintaprogramunkba, hogy legyen hova ugrani. Amennyiben folytatódna a program, minden esetben ezt a megoldást kellene választanunk.

A 40-es sorban a STOP és az 50-es sorban az END utasításnak azonos a szerepük. Megállítják a program futását. A STOP utasítás hatására a számítógép azt is kiírja, hogy a program melyik soron állt meg

BREAK IN 30

A programozásban megkülönböztetjük a program *fizikai* végétől, a *logikai* végét. A legnagyobb sorszámú utasítássor lesz a program fizikai vége. A program logikai végén értjük azt, ahol a program futása befejeződik.

● Értelmezzük a következő programot, aminek az algoritmusát a 17. ábra mutatja! A programban egy utasítás dönti el, hogy X értékétől függően mi történjen. Ez az utasítás az ON–GOTO.

```
10   REM OSZTALYZAT-2
20   INPUT "OSZTALYZAT",X
30   ON X GOTO 50,60,70,80
40   X$="JELES":GOTO 90
50   X$="ELEGTELEN":GOTO 90
60   X$="ELEGSEGES":GOTO 90
70   X$="KOZEPES":GOTO 90
80   X$="JO"
90   PRINT X$
```

Az ON–GOTO utasítással megadjuk, hogy X értékétől függően, hol folytatódjon a program. Akkor, ha az X értéke nagyobb, mint amennyi sorszámot megadtunk az utasításban, vagy 0, akkor a következő utasítássorral folytatódik a program. Ezt használtuk ki a JELES kiírásánál. $X < 0$ esetén hibát jelez a rendszer.

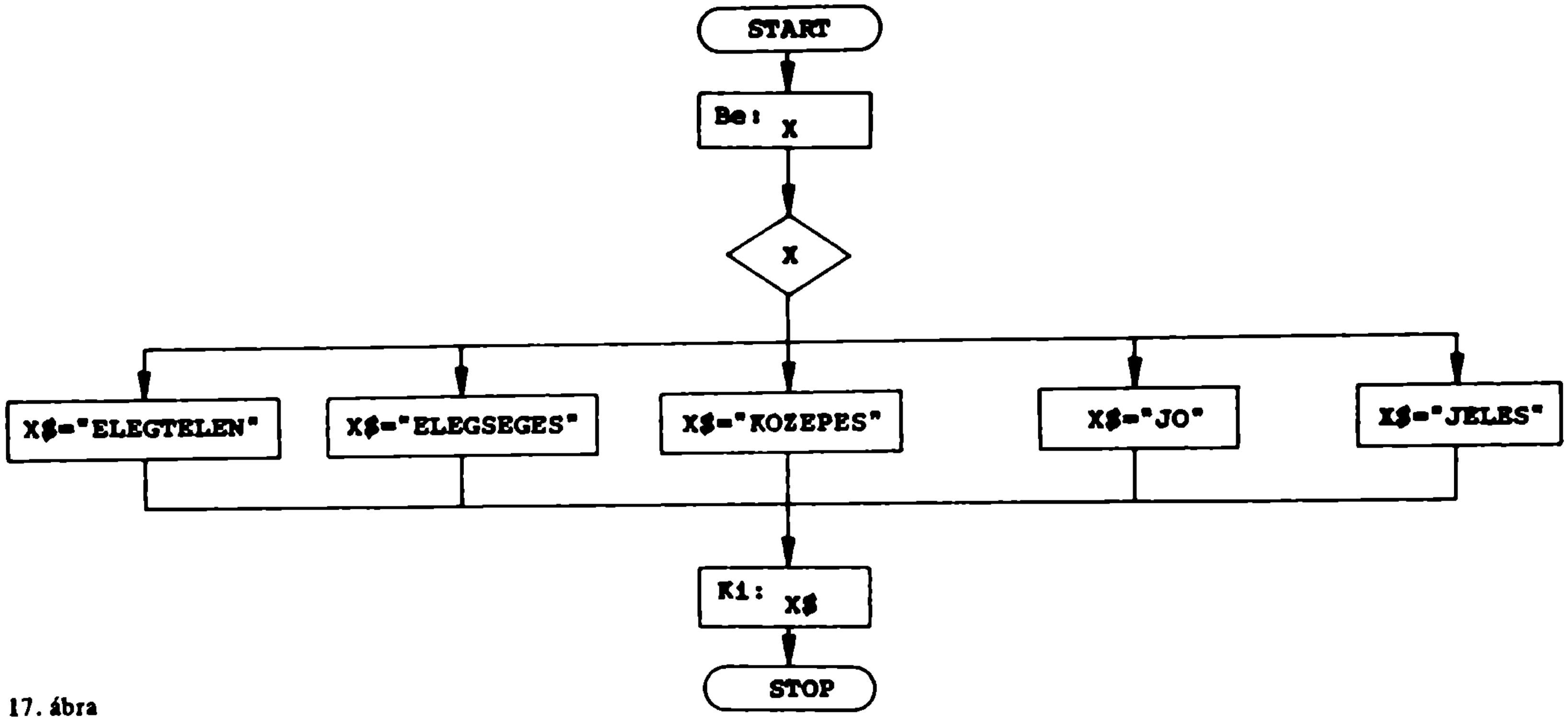
● Írassuk ki a tanulmányi átlagot betűvel, ha a rendűségi határok: 1,75; 2,75; 3,75 és 4,75!

Elég az előző program 30-es sorát átírni:

```
30   ON  X+0.25 GOTO 50,60,70,80
```

Az ON–GOTO utasításban tetszőleges aritmetikai kifejezés szerepelhet. Ennek egészértéke határozza meg, hogy hol folytatódjon a program. A kifejezés értéke nem lehet negatív, mert akkor hibát jelez a rendszer.

Az *IF utasítást* tekinthetjük *logikai*, az *ON utasítást aritmetikai* feltételes vezérlő utasításnak. Az ON utasítást *többirányú elágaztató* utasításnak is nevezik.


START
Be: X
X
X$="ELEGTELEN"
X$="ELEGSEGES"
X$="KOZEPES"
X$="JO"
X$="JELES"
Ki: X$
STOP

17. ábra

Alakja:
ON aritmetikai kifejezés GOTO sorszámlista
jelentése: az aritmetikai kifejezés egész értéke megmutatja, hogy a sorszámlista hányadik elemét tekintsük a következő végrehajtandó utasítássornak.

- Akkor, ha a kifejezés egész értéke 0 vagy nagyobb, mint amennyi elemből áll a sorszámlista, akkor a következő utasítássorral folytatódik a program.
- Negatív érték esetén hibajelzéssel megáll a program.

● Készítsünk programot, amivel a képernyõn egy karaktert mozgathatunk a J, B, L, F billentyűk segítségével! Akkor, ha a J billentyűt nyomjuk le, a karakter az előző karakter mellett jobbra, ha a B-t nyomjuk le, balra jelenjen meg. Az F és az L billentyű lenyomására felette, ill. alatta jelenjen meg!

A PRINT@(olvasd PRINT egységár v. „kukac") utasítással a képernyõ tetszõleges karakterpozíciójába állíthatjuk a kurzort, és ettől a pozíciótól kezdõdõen történik a kiírás.

A *képernyõ karakterpozícióinak sorszámozása* a képernyő bal felső sarkától 0-val kezdõdõen, sorfolytonosan történik. Ez azt jelenti, hogy az első sor utolsó karakterpozíciója a 63., mivel egy sorba 64 karaktert írhatunk, így a második sor első karakterpozíciója lesz a 64. A képernyõn 16 sor jeleníthető meg, tehát összesen 16·64 = 1024 karakter írható a képernyőre. Az utolsó karakterpozíció az, amelyik a képernyő jobb alsó sarkában van. Ez a 1023. lesz.

A következõ programmal a képernyő minden pontjába elhelyezzük a * karaktert. Mielőtt a csillagok kiíratását elkezdenénk, *töröljük a képernyõt* a CLS utasítással.

```
10  REM CSILLAGOK
20  CLS:REM KEPERNYO TORLES
30  I=0
40  PRINT@ I,"*",
50  I=I+1:IF I <=1023 THEN 40
```

A PRINT @I, "*" utasítással a kurzort a képernyõ I. karakterpozíciójába állítjuk, ahová kiíratjuk a karaktert. Az I értékét ciklussal változtatjuk.

A PRINT @ utasításban a képernyõ karakterpozícióját kijelölõ aritmetikai kifejezés után vesszõt írunk, és ezután következik a kiíratási lista. Ezzel az utasítással a képernyõ bármely pozíciójába tudunk írni.

A lenyomott billentyűt az INKEY$ függvény érzékeli. Az INKEY$ függvény értéke az a karakter lesz, amelyik billentyűt lenyomva tartjuk a függvény kihívásakor. Az INKEY$ függvénnyel beolvasott karakter nem jelenik meg a képernyõn.

Az eredeti feladat programja tehát a következő lesz:

```
10  REM KARAKTER MOZGATASA
20  CLS:I=7:J=32
30  IT=I:JT=J:PRINT@ ,I * 64+J,"X";
40  X$=INKEY$:IFX$=""THEN 40
50  IF X$="J" THEN J=J+1:GOTO 90
60  IF X$="B" THEN J=J-1:GOTO 90
70  IF X$="F" THEN I=I-1:GOTO 90
80  IF X$="L" THEN I=I+1
```

```
 90  PRINT@,IT*64+JT," ",
100  IF I*64+J<0 THEN END
110  IF I*64+J<1024 THEN 30
```

Az INKEY$ függvénnyel beolvassuk a karaktert, és a karaktertől függően változtatjuk az I, ill. a J változók értékeit, amivel a képernyő sorát, ill. oszlopát adjuk meg, ahová a következő *-ot íratjuk ki. Az IT és JT változókkal adott karakterpozícióba szóköz karaktert írunk, amivel töröljük a * karaktert.

● Készítsünk programot, amivel úgy jeleníthetjük meg a BASIC szót a képernyő legfelső sorában, hogy a BASIC szóból mindig eggyel több betűt lássunk a képernyő jobb oldalán, és az így kialakított szó fusson végig a képernyőn, a képernyő bal oldalára!

Karakterláncokat kell kiíratnunk, ami először egy karakterből áll, legvégül ötből. A karakterek összefűzését karakterenként végezzük. Ezeket a műveleteket egymás utáni utasításssorokban hajtjuk végre. Az ON–GOTO utasítással megoldhatjuk azt, hogy mindig annyi karakter alkossa a karakterláncot, amennyit ki akarunk írni. Akkor, amikor már minden karaktert kiírtunk, csak a „lemaradó" karakterek törléséről kell gondoskodni (60-as sor). A karakterláncokról és a karakterláncokkal végezhető műveletekről később még részletesen szó lesz. Az ott leírtak ismeretében a feladat egyszerűbben is megoldható.

```
 10  REM KEPUJSAG
 20  I=1
 30  CLS
 40  X$=""
 50  ON I GOTO 110,100,90,80,70
 60  X$=" "+X$
 70  X$="C"+X$
 80  X$="I" X$
 90  X$="S"+X$
100  X$="A"+X$
110  X$="B"+X$
120  PRINT@ 64-I,X$
130  I=I+1 IF I<=64 THEN 40
```

Kérdések és feladatok

1. Milyen vezérlő utasításokat ismerünk? Hogyan csoportosíthatjuk azokat?
2. Hogyan lehet a futó programot leállítani?

3. Írassuk ki programmal a 200-nál kisebb pozitív páratlan számokat csökkenő sorrendben!

4. Írjunk programot, amivel kiírathatjuk a 2^n+1 alakú egész számokat! Csak pontos értékek jelenjenek meg a képemyőn!

5. Mit csinál a következő program?

```
10  INPUT  A
20  IF  A THEN PRINT 1/A
```

Magyarázd meg!

6. Elhagyhatjuk-e a: GOTO 70, STOP és END utasitásokat az osztályzatokat kiíró programban? Miért?

7. A KARAKTER MOZGATASA programban oldjuk meg azt, hogy mindig csak egy karakter legyen a képernyőn!

8. Értelmezzük a következő programot!

```
10  X=1
20  PRINT  X
30  Q$=INKEY$:IF  Q$=""THEN 30
40  X=X+1:IF  Q$<>"V"THEN 20
```

9. Értelmezzük a következő folyamatábrát (18. ábra), és írjuk meg a programját!

18. ábra

10. Írjunk programot:

a) az elsőfokú kétismeretlenes egyenletrendszer megoldására;

b) a másodfokú egyenlet megoldására;

c) a háromszög területének és kerületének kiszámítására, ha adottak az oldalak;

d) számtani és mértani sorozatok első n elemének az előállítására;

e) n szám összegének és szorzatának meghatározásához;

f) amivel teleírhatjuk a képernyőt a SZAMÍTÓGÉP szóval!

11. Oldjuk meg kétféleképpen nyolc számjegynél kevesebb számjegyből álló bármilyen bináris szám decimális számmá alakítását!

a) Szervezzünk ciklust!

b) Használjuk az ON–GOTO utasítást!

12. Mi történik a következő program hatására?

```
10  I=20
20  PRINT@15*64+I,"*"
30  I=I+1:IF I<40 THEN 20
40  I=40
50  PRINT@15*64+I," ",
60  I=I-1:IF I>20 THEN 50
70  IF INKEY$="" THEN 10
```

VIZSGÁLAT

Egyszerű példán, a szám abszolút értékét előállító program néhány változatával mutatjuk be, hogy ugyanazt az eredményt hányféleképpen kaphatjuk meg, ha különböző relációjeleket használunk az IF utasításban.

```
10  INPUT X
20  IF X<0 THEN X=-X
30  PRINT X
```

```
10  INPUT X
20  IF X>0 THEN 30 ELSE X=-X
30  PRINT X
```

```
10  INPUT X
20  IF X>0 THEN 40
30  X=-X
40  PRINT X
```

A vizsgálat ügyes megválasztásával rövidebbé és áttekinthetőbbé tehetjük programjainkat.

● Írjunk programot N darab szám összegének és szorzatának meghatározására!

A *ciklusmag* N-szeres ismétlését az előző fejezetben bemutatott formában oldjuk meg.

– A ciklusmagban helyezzük el a számok beolvasását végző, összegüket és szorzatukat képző utasításokat.

– A ciklusmag része lesz *a ciklusváltozó értékének módosítása* is, az I = I+1 utasítás.

– A ciklusba lépés előtt gondoskodnunk kell az összeg és a szorzat értékét tartalmazó *változók kezdőértékéről.*

A számok összegét az S változó értéke adja, ennek kezdőértéke 0. A P változóban a számok szorzatát tároljuk, amelyiknek 1 kezdőértéket kell adni.

```
10  REM  OSSZEG, SZORZAT
20  INPUT "N",N
30  S=0:P=1:I=1
40  INPUT  X
50  S=S+X:P=P*X
60  I=I+1:IF I<=N THEN 40
70  PRINT "OSSZEG",S,"SZORZAT",P
```

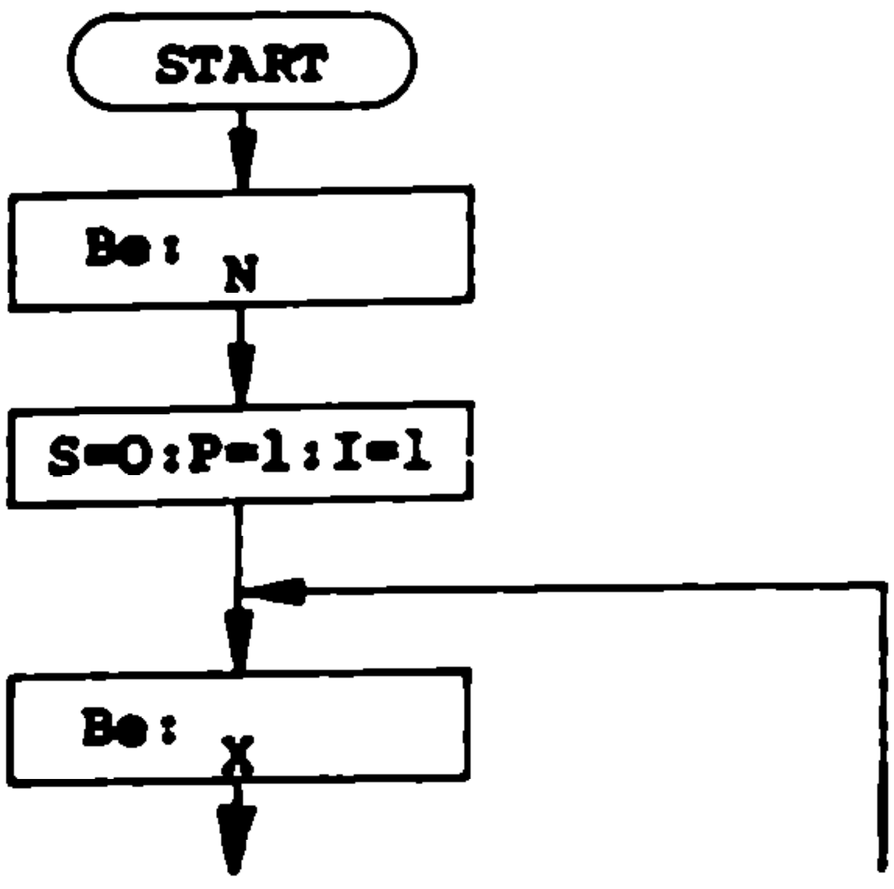
A ciklusváltozó értékét más relációval is vizsgálhatjuk, de ekkor más helyre kell írnunk a ciklusváltozó értékét módosító utasítást, és meg kell változtatnunk az IF utasítás szerkezetét is. Néhány változatot mutatunk be. Csak a program végét írjuk le, ahol módosítottuk az eredeti programot.

```
60  IF I<N THEN I=I+1:GOTO 40
70  PRINT "OSSZEG",S,"SZORZAT",P
```

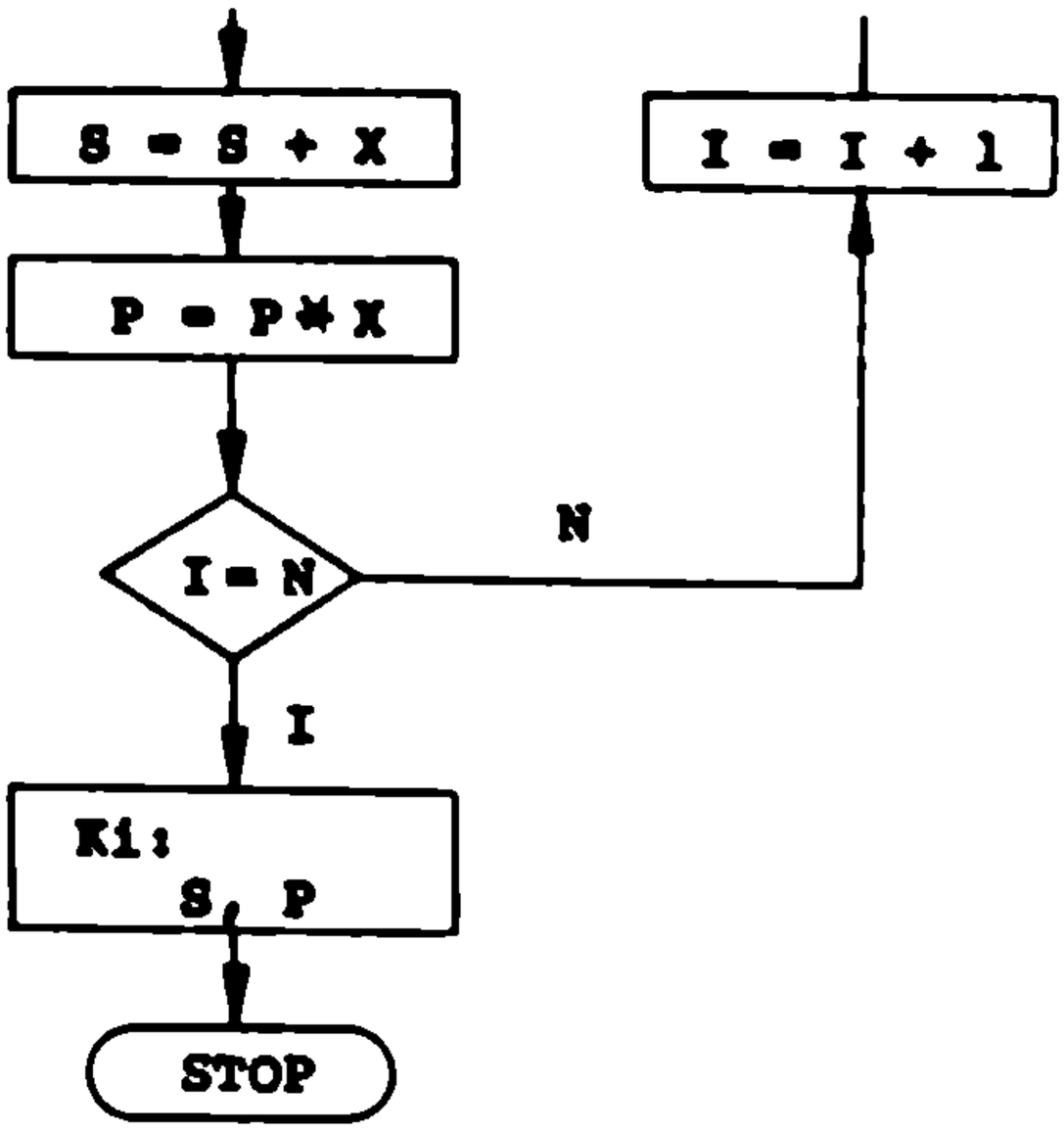
```
60  IF I>= N THEN 70 ELSE I =I+1:GOTO 40
70  PRINT "OSSZEG",S,"SZORZAT",P
```

```
60  IF I=N THEN 80
70  I=I+1:GOTO 40
80  PRINT "OSSZEG",S"SZORZAT",P
```

Az utóbbi program folyamatábrája a következő (19. ábra):



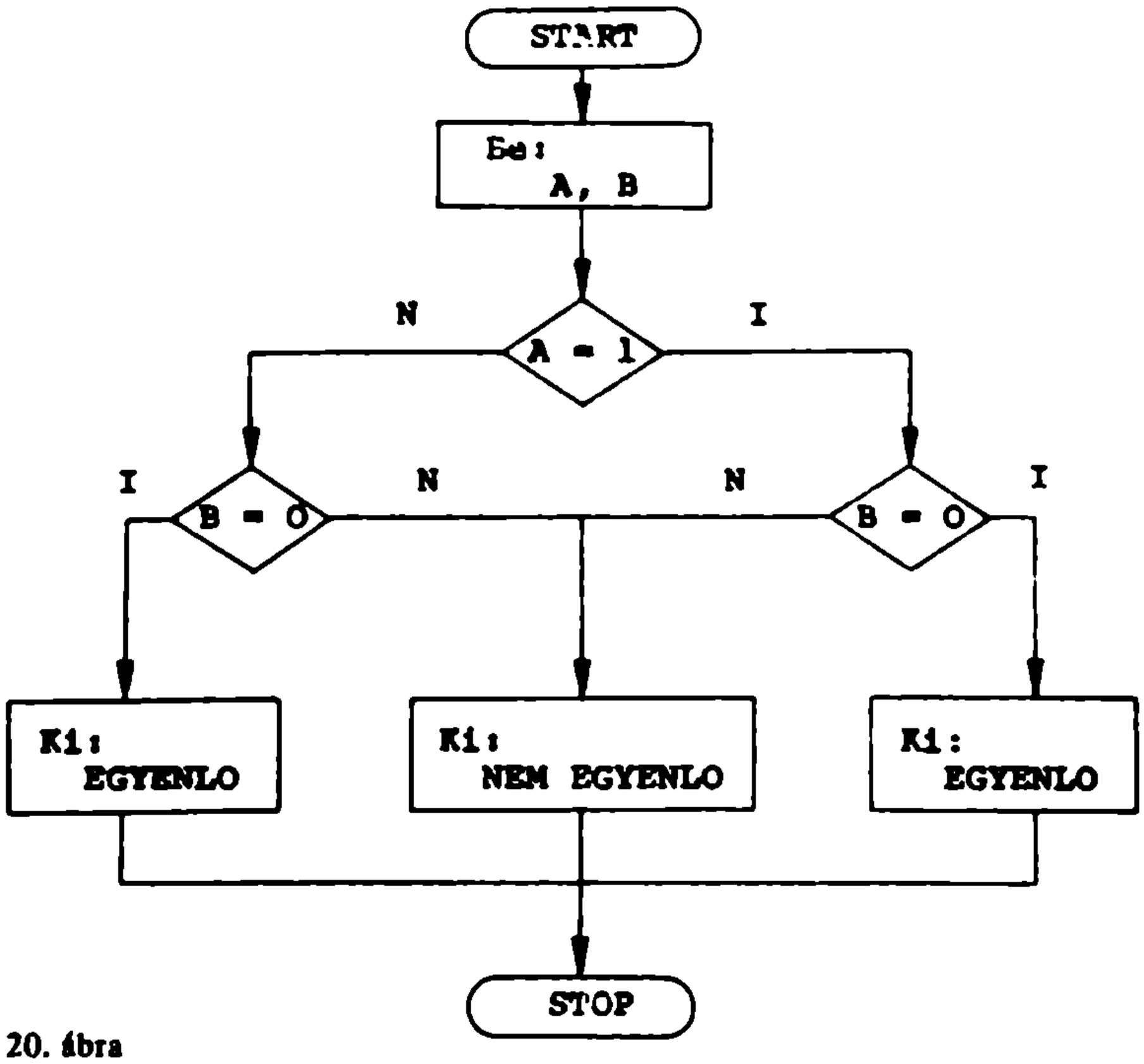
19. ábra



19. ábra

● Két tanuló azt szeretné játszani, hogy ha 0-t, vagy 1-et begépelnek, akkor a számítógép döntse el, hogy a begépelt számok azonosak-e. Írjunk programot a játékhoz!

A 20. ábrán látható a folyamatábra egy lehetséges megoldását adja a feladatnak.



20. ábra

```
10   REM FEJ, VAGY IRAS
20   INPUT "ELSO JATEKOS";A
30   INPUT "MASODIK JATEKOS";B
```

```
40   IF A=1 THEN 70
50   IF B=0 THEN PRINT "EGYENLO",:END
60   GOTO 80
70   IF B=1 THEN PRINT "EGYENLO",:END
80   PRINT "NEM EGYENLO"
```

A vizsgálatok és a kiíratás az ELSE használatával:

```
40   IF A=1 THEN 60
50   IF B=0 THEN PRINT "EGYENLO",:END
          ELSE PRINT "NEM EGYENLO",:END
60   IF B=1 THEN PRINT "EGYENLO"
          ELSE PRINT "NEM EGYENLO"
```

A THEN és az ELSE szavak után bármilyen utasítást írhatunk, tehát IF utasítást is.

```
40   IF A=1 THEN IF B=1 THEN PRINT
        "EGYENLO",:END ELSE 60
50   IF B=0 THEN PRINT "EGYENLO",:END
60   PRINT "NEM EGYENLO"
```

A 40-es sorban a B = 1 reláció kiértékelésére csak akkor kerül sor, ha az A = 1 állítás IGAZ. Akkor, ha HAMIS, az 50-es utasítássor következik.

Két szám négyzetösszege akkor és csak akkor 0, ha mindkét szám 0, és két szám szorzata akkor 0, ha legalább az egyik tényező 0. Ezeket a matematikai törvényszerűségeket használjuk fel a következő programban.

```
40   IF A↑2+B↑2=0 THEN PRINT "EGYENLO",
                                  :END
50   IF A*B=0 THEN PRINT "NEM EGYENLO"
               ELSE PRINT "EGYENLO"
```

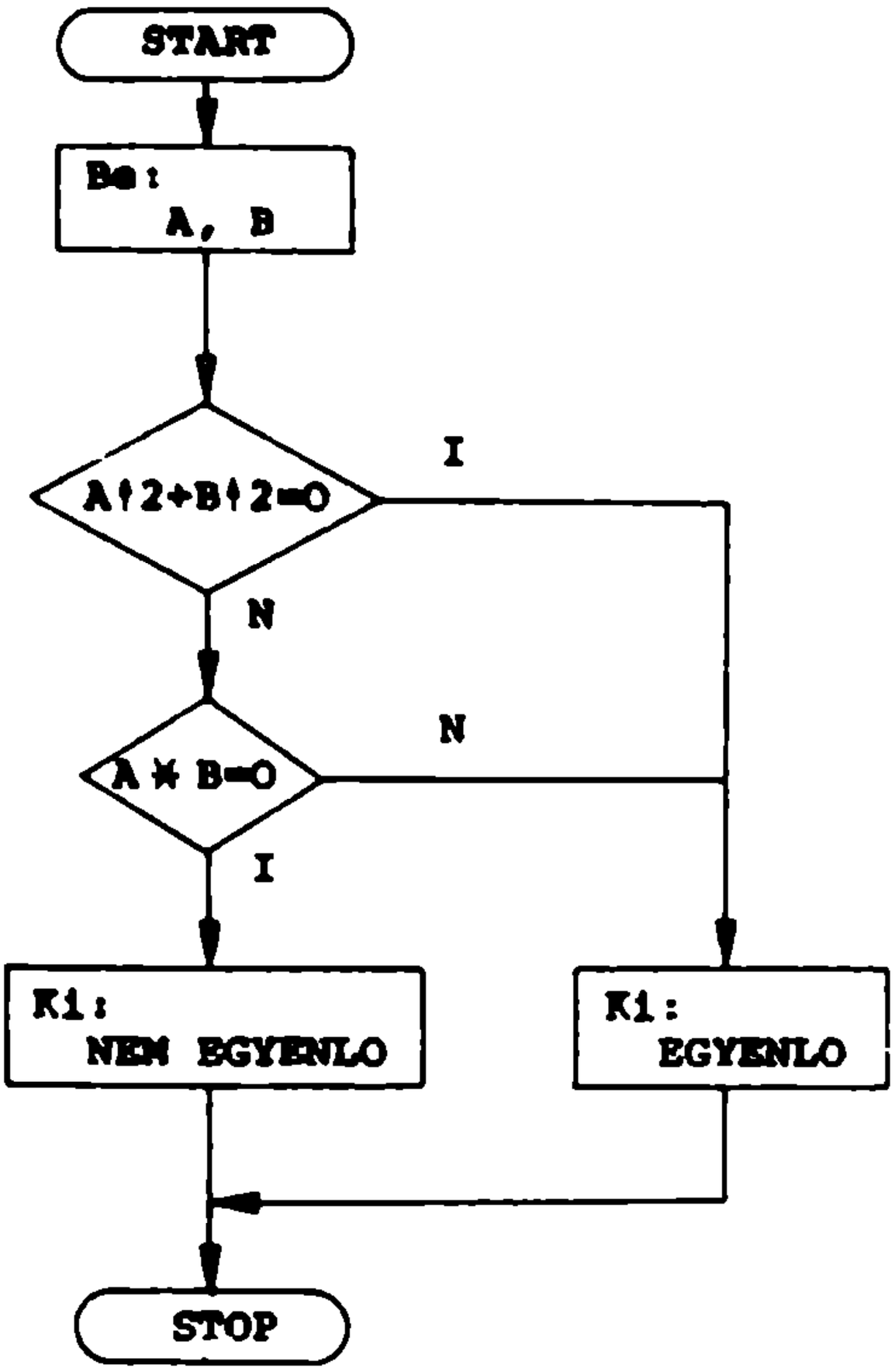
Ekkor a program folyamatábrája a 21. ábra szerinti.

Oldjuk meg a feladatot az ON–GOTO utasítással is! Használjuk ki azt a követelményt, hogy a változók értéke csak 0 vagy 1 lehet. Összegük ezért 0, 1 vagy 2. Vegyük észre, hogy a feladatot megoldhatjuk az ON–GOTO utasítással is.

```
40   ON A+B+1 GOTO50,60,50
50   PRINT "EGYENLO",END
60   PRINT "NEM EGYENLO"
```

Egyszerűbben megtervezhető a program, ha az állítást *logikai műveletekkel* fogalmazzuk meg. Háromféle logikai műveletet alkalmazhatunk:

```
NOT   a tagadas,
AND   az ÉS
 OR   a VAGY logikai műveletet jelent
```



21. ábra

Összetett logikai kifejezésben szereplő logikai műveletek kiértékelése is ebben a sorrendben történik. Oldjuk meg az előző feladatot úgy is, hogy logikai műveletekkel fejezzük ki az állítást!

```
40  IF (A=1 AND B=1) OR (A=0 AND B=0)
        THEN PRINT "EGYENLO" ELSE
        PRINT "NEM  EGYENLO"
```

A zárójelet elhagyhatjuk, mivel először az AND műveletet hajtja végre a számítógép.

A programban használt logikai kifejezéssel az *AZONOSSÁG logikai műveletet* valósítottuk meg. Ennek a műveletnek az értéke akkor IGAZ, ha az állítások, amiken a műveleteket végezzük, azonos értékűek: mindkettő IGAZ vagy mindkettő HAMIS. Igazságtáblázata a következő (22. ábra):

A	B	E
1	1	1
1	0	0
0	1	0
0	0	1

22. ábra

● Írjunk programot az elsőfokú kétismeretlenes egyenletrendszer megoldására! A feladat megoldását azzal kell kezdeni, hogy megoldjuk az

$$ax+by = c$$
$$dy+ey = f$$

elsőfokú egyenletrendszert!

Az egyenletrendszer gyökei:

$$X = \frac{ce-bf}{ae-bd} \qquad Y = \frac{af-dc}{ae-bd},$$

ha $ae-bd \neq 0$.

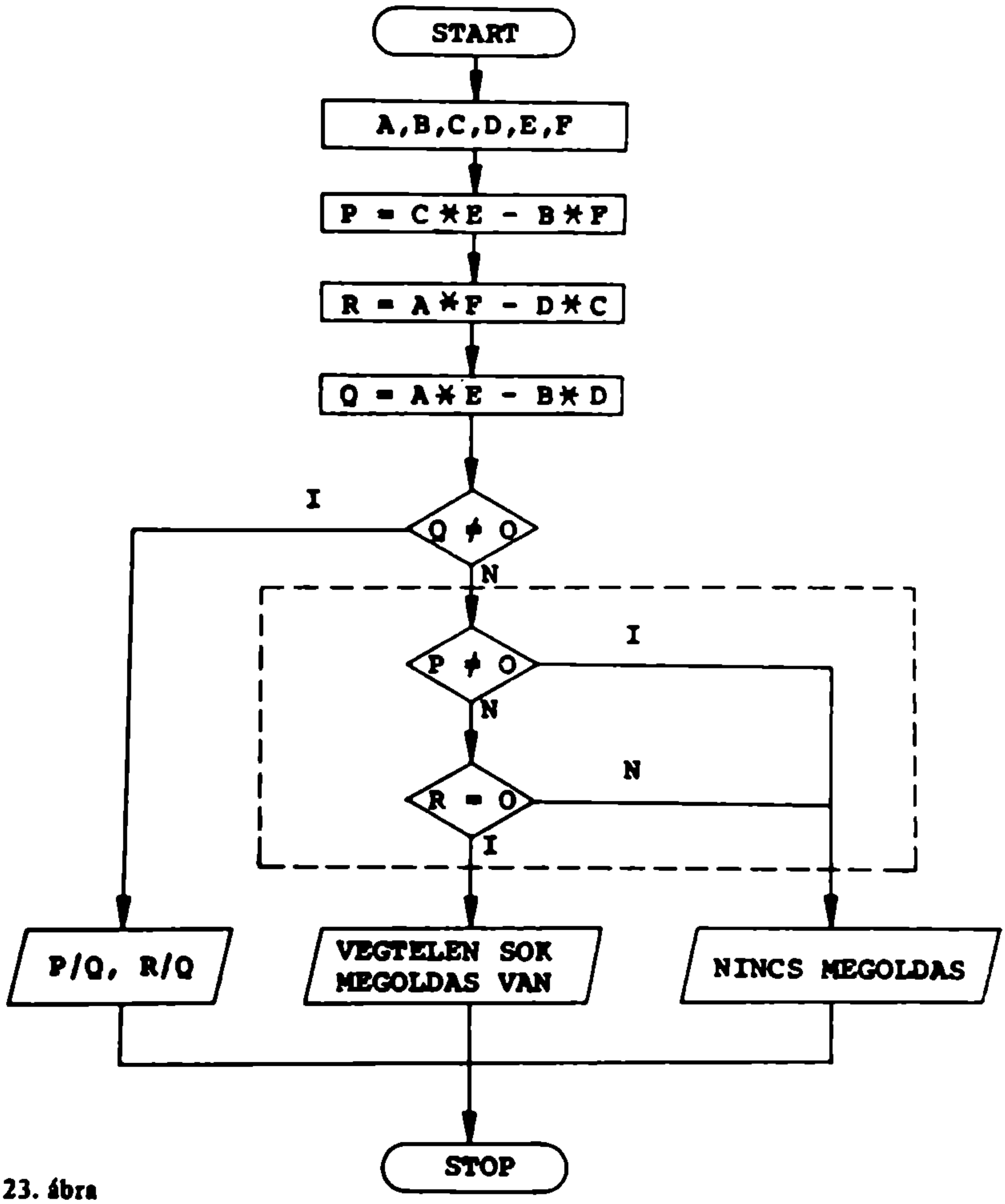
Akkor, ha $ae-bd = 0$ és

– $af-cd \neq 0$ V $ce-bf \neq 0$, nincs megoldása az egyenletrendszernek;

– végtelen sok megoldás van, ha

$af-cd = 0 \wedge ce-bf = 0$.

Az algoritmusunk az előzőek szerint a következő (23. ábra):



23. ábra

```
10  REM ELSOFOKU EGYENLETRENDSZER
20  INPUT "EGYUTTHATOK,KONSTANSOK",A,B,C,D,E,F
30  P=C*E-B*F:R=A*F-D*C:Q=A*E-B*D
40  IF Q<>0 THEN PRINT P/Q,R/Q:STOP
50  IF P<>0 THEN 70
60  IF R=0 THEN PRINT "VEGTELEN SOK
          MEGOLDAS VAN":STOP
70  PRINT "NINCS MEGOLDAS"
```

A 23. ábrán ábrázolt folyamatábra szaggatott vonallal jelzett részét egy vizsgálattal a következőképpen oldhatjuk meg:

```
50  IF P↑2+R↑2<>0 THEN PRINT "NINCS
                    MEGOLDAS":STOP
60  PRINT "VEGTELEN SOK MEGOLDAS VAN"
```

A feladat diszkussziójának leírásában a matematikában megszokott logikai műveletekkel adtuk meg a feltételeket. A programban ezeket a feltételeket fogalmazzuk meg a BASIC nyelv szabályai szerint.

Logikai műveletekkel a vizsgálat:

```
50  IF A<>0 OR B<>0 THEN PRINT "NINCS
                    MEGOLDAS":STOP
60  PRINT "VEGTELEN SOK MEGOLDAS VAN"
```

Itt a VAGY logikai műveletet használtuk. Az ES művelettel a megoldásunk a következő:

```
50  IF A=0 AND B=0 THEN PRINT
        "VEGTELEN SOK MEGOLDAS":STOP
60  PRINT "NINCS MEGOLDAS"
```

Kérdések és feladatok

1. Egészítsük ki az OSSZEG, SZORZAT programot! Csak a 100-nál kisebb pozitív számokat összegezzük!

2. Írjunk programot az $(x-a)(x+b) = 0$ alakú egyenlet megoldására!

3. Hogyan oldanánk meg a programban a KIZARO VAGY logikai műveletet?

4. Ellenőrizzük programmal a de Morgan-féle azonosságot:

$\overline{a \wedge b} = \bar{a} \vee \bar{b}$!

5. Készítsünk programot a következő függvény helyettesítési értékeinek kiszámítására:

$$x \to \begin{cases} x^2-3, & \text{ha } x < -3 \\ 2, & \text{ha } -3 \leqslant x \leqslant 2 \\ -x^2+3, & \text{ha } x > 2 \end{cases}$$

6. Írjunk programot, amivel meghatározhatjuk

a) két pont kölcsönös helyzetét a számegyenesen;

b) egy pont és egy egyenes;

c) egy pont és egy kör kölcsönös helyzetét a síkban!

Amennyiben két pont távolsága kisebb, mint 0,0001, a két pontot tekintsük illeszkedőnek.

7. Írjunk programot, amivel kiírhatjuk a hét napjainak nevét, ha megadjuk a napok sorszámát!

ADATOK A PROGRAMBAN

• Írassuk ki programmal a hét napjainak nevét, ha megadjuk, hogy hányadik napra gondoltunk!

```
10  REM HET NAPJAI
20  INPUT "HANYADIK NAP",N
30  IF N=1 THEN PRINT "HETFO":END
40  IF N=2 THEN PRINT "KEDD":END
50  IF N=3 THEN PRINT "SZERDA":END
60  IF N=4 THEN PRINT "CSUTORTOK":END
70  IF N=5 THEN PRINT "PENTEK":END
80  IF N=6 THEN PRINT "SZOMBAT":END
90  PRINT "VASARNAP"
```

A hét napjainak nevét tekinthetjük adatoknak. Ezek közül kell kikeresnünk a megfelelő sorszámút. A kikereséshez csaknem annyi IF utasítást használtunk, amennyi adat van. Ez a megoldás már ennyi adatnál is kényelmetlen.

Helyezzük el a hét napjainak nevét mint adatokat a DATA utasítással a programba. Az így megadott adatokhoz a READ utasítással tudunk hozzáférni.

A programban a READ X$ utasítással olvassuk be, a legkisebb sorszámú DATA utasítással kezdődően az adatokat. Amennyiben egy DATA utasítás után nincs több adat, akkor a beolvasás a következő DATA utasítás adatainak beolvasásával folytatódik.

A DATA utasításban az adatokat vesszővel választjuk el egymástól. Egy DATA utasításban szereplő adatok alkotják az *adatlistát*.

A READ X$ utasítást annyiszor hajtja végre a program, ahányadik nap nevét kérjük. A beolvasás elkezdődik a legkisebb sorszámú adatlistából és sorba beolvassa a program az adatokat, míg a kérdéses nap nevéhez nem ér. Amikor kilépünk a ciklusból az X$ értéke a keresett nap neve lesz.

```
10  REM KIKERESES DATA-BOL
20  INPUT "HANYADIK NAP",N
30  I=1
40  READ X$
50  I=I+1:IF I<=N THEN 40
60  PRINT "EZ A NAP",X$
70  DATA HETFO,KEDD,SZERDA
80  DATA CSUTORTOK,PENTEK
90  DATA SZOMBAT,VASARNAP
```

Ezzel a programmal tulajdonképpen bármilyen adatsorból kikereshetjük az N. elemet. A DATA adatlistát kell csak átírni.

Egy adatsor adott elemének sorszámát is hasonlóképpen határozhatjuk meg. Nézzünk erre egy példát!

● Egy atlétikai verseny résztvevőinek nevét, a rajtszámuk növekvő sorrendjében a DATA adatmezőbe beírjuk.

A *DATA adatmezőn* a DATA adatlistákban szereplő adatok összességét értjük.

Írjunk programot, amivel a versenyzők neve alapján meghatározhatjuk a versenyző rajtszámát! Az adatmezőből egymás után beolvassuk a versenyzők neveit, és számláljuk, hogy a megadott név az adatmező hányadik eleme lesz. Ez lesz a versenyző rajtszáma, ha a rajtszámok egyesével növekednek.

A ki nem osztott rajtszámoknak megfelelő helyre * karaktereket írunk.

```
10  REM RAJTSZAMOK
20  INPUT "A VERSENYZO NEVE"; X$
30  I=1
40  READ Y$
50  IF X$=Y$ THEN PRINT "A RAJTSZAM",I:END
60  I=I+1:IF I<8 THEN 40
70  PRINT "NEM TALALOM"
80  DATA KIS,KOVACS,VARGA
90  DATA *,NAGY,*,SZABO
```

A programban hét adatot adtunk meg. Gondoskodtunk arról, hogy csak ennyi adatot olvashasson be a program. Akkor, ha olyan nevet adunk meg, ami nem szerepel az adatok között, kiírja, hogy NEM TALALOM.

A vizsgálatra azért van szükség, mert ha a READ utasítással akkor is akarunk adatot beolvasni, amikor már az *adatmező végére értünk*, akkor a számítógép hibát jelez.

● Készítsünk „szótár" programot! A program adja meg a BASIC utasításszavak magyar jelentését! Az utasításszavakat és azok jelentését beírjuk a DATA adatlistákba. Innen kell kikeresnünk a megfelelőt.

Most is gondolnunk kell arra, hogy a READ utasítással a beolvasást csak annyiszor kíséreljük meg, amennyi adatunk van. Előfordulhat, hogy olyan utasításszót adunk meg, amelyik nem szerepel a megadott adatok között. Ez természetesen csak akkor derül ki, ha az adatmező adatait végigvizsgáltuk. Azt, hogy *mikor értünk az adatmező végére*, többféleképpen megadhatjuk.

Ha *ismerjük az adatok számát*, akkor a beolvasást és a vizsgálatot egy olyan ciklusban helyezzük el, amivel csak annyi adatot olvashatunk be, amennyi van.

```
10  N=6:I=1
20  INPUT "AZ UTASITASSZO",X$
30  READ U$,J$
40  IF U$=X$ THEN 70
50  I=I+1:IF I<N THEN 30
60  PRINT "NEM TALALOM": STOP
70  PRINT U$; "JELENTESE",J$
100 DATA READ,OLVASS,DATA,ADATOK
110 DATA IF,HA,STOP,ALLJ,THEN,AKKOR
```

Az előző megoldás annyiból nem a legszerencsésebb, hogy új adatok beírásakor, a szókészlet bővítése esetén, változtatnunk kell az N értékét is.

Jelöljük az adatmező végét valamilyen jellel, ami nem fordulhat elő az adatok között. Ennek típusában meg kell egyeznie a többi adattal.

Programunkban egy utasítással két adatot olvasunk be, ezért két karakterlánccal jelezzük az adatmező végét.

```
10  INPUT "AZ UTASITASSZO",X$
20  READ U$,J$
30  IF U$="* * *" THEN 60
40  IF U$<>X$ THEN 20
50  PRINT U$, "JELENTESE",J$:END
60  PRINT "NEM TALALOM"
100 DATA READ,OLVASS,DATA,ADATOK
110 DATA IF,HA,STOP,ALLJ,THEN,AKKOR
200 DATA * * *, * * *
```

● Számítsuk ki, hogy egy adott munkabér esetén mennyi nyugdíjjárulékot kell fizetni!

A nyugdíjjárulék a bér egy bizonyos százaléka, ami a bértől függ. Ismernünk kell a bérhatárokat, és a bérhatárokhoz tartozó százalékokat, hogy a számítást elvégezhessük:

1800-ig 3 %,

2300-ig 4 %,

3000-ig 5 % stb.

Az adatokat DATA-ban helyezzük el, ahonnan sorban kiolvassuk, és összehasonlítjuk a megadott bérrel. Célszerű a vizsgálatot a legmagasabb bérhatártól kezdeni. Ekkor a program egyszerűbb.

```
10  REM NYUGDIJJARULEK
20  DATA 24000,15,12000,12,10000,13,8000,12,7000,11
30  DATA 6000,9,5000,8,4000,7,3000,6,2300,5,1800,4
35  DATA 1000,3
40  INPUT "A BER",B
50  READ H,S:REM HATAR,SZAZALEK
60  IF B<H THEN 50
70  PRINT "NYUGDIJJARULEK",B*S/100
```

A DATA adatlista a programban bárhol elhelyezhető. Általában a programtól elkülönítve, az elejére vagy a végére írjuk.

● Adott N darab számjegy. Írassuk ki azokat betűvel!

A számjegyek nevének kikeresésekor valameddig eljutunk az olvasással az adatmezőben, mire megtaláljuk a megfelelő adatot. A következő számjegy kikeresését ismét az adatmező elejéről kell kezdenününk, ezért használnunk kell a RESTORE utasítást.

A RESTORE utasítás hatására, a következő READ utasítás az adatmező elejéről, tehát a legkisebb sorszámú adatlista elejéről olvassa be az adatot.

```
 10  REM SZAMJEGYEK BETUVEL
 20  INPUT "SZAMJEGYEK SZAMA",N
 30  I=1:ON ERROR GOTO 200
 40  INPUT "A SZAMBEGY",Y,:RESTORE
 50  J=0
 60  READ Z$
 70  J=J+1:IF J<=Y THEN 60
 80  PRINT Y,"-",Z$
 90  I=I+1:IF I<=N THEN 40
100  END
110  DATA NULLA,EGY,KETTO,HAROM,NEGY,OT
120  DATA HAT,HET,NYOLC,KILENC,
200  PRINT "ILYEN SZAMJEGY NINCS"
210  RESUME 40
```

Ebben a programban *két ciklust ágyaztunk egymásba.* A külső ciklus a betűket számolja, a belső a keresést végzi.

A RESTORE utasítást akkor adjuk ki, amikor elkezdődik a keresés, amikor belépünk a belső ciklusba. A kikeresés tehát az adatmező elejéről kezdődik.

● A * karakterekkel jelenítsük meg a képernyőn az A betűt!

A számítógép a betűket a képernyő pontjaiból állítja össze. Minden betűt egy 5X7-es pontmatrix világító pontjai alkotják. Az A betű pontmatrixa a következő (24. ábra):

16	8	4	2	1
		X		
	X		X	
X				X
X				X
X	X	X	X	X
X				X
X				X

24. ábra

A pontmatrixok megjelenítésének legegyszerűbb módja, ha a matrix minden pontjának megadjuk az állapotát. Programunkban 0-val jelezzük, ha a pont nem világít, 1-gyel, ha világít. Ennek megfelelően írjuk ki a * karaktereket.

```
10  REM BETUK-1
20  CLS
30  I=1
40  J=5
50   READ M
60  IF M=1 THEN PRINT@(5+I)*64+10+J," *",
70  J=J-1:IF J>0 THEN 50
80  I=I+1:IF I<8 THEN 40
 90  DATA 0,0,1,0,0
100  DATA 0,1,0,1,0
110  DATA 1,0,0,0,1
120  DATA 1,0,0,0,1
130  DATA 1,1,1,1,1
140  DATA 1,0,0,0,1
150  DATA 1,0,0,0,1
```

A betűket mindenképpen meg kell terveznünk. Az így készített ábrák alapján egyszerűbb úgy tekinteni a sorokat, mint bináris számokat és azokat decimális alakban megadni a DATA-ban, mint megadni a pontok állapotát külön-külön, ahogyan azt az előző programban tettük.

A *programmal alakítjuk át* a decimális számot binárissá, és az így kapott bináris számjegyek határozzák meg, hogy kiírjuk-e a * karaktert vagy nem. Az átalakítás algoritmusa

10:2 = 5	5:2 = 2	2:2 = 1	1:2 = 0	0:2 = 0
0	1	0	1	0

A maradékok adják a mátrix második sorának pontjait.

```
10  REM BETUK-2
20  CLS
30  I=1
40  READ X
50  J=5
60  H%=X/2:M=X-2*H%
 70  IF M=1 THEN PRINT@(5+I)*64+10+J,"*",
 80  X=H%
 90  J=J-1:IF J>0 THEN 60
100  I=I+1:IF I<8 THEN 40
110  DATA 4,10,17,17,31,17,17
```

Kérdések és feladatok

1. Gondoljuk végig, hogy hogyan oldanánk meg a fejezetben található programokat a READ és a DATA utasítások ismerete nélkül!

2. Mi a RESTORE utasítás hatása?

3. Figyelembe veszi-e a rendszer, ha a karakterlánc-konstansokban szóköz karaktereket írunk!

4. Helyes-e a következő adatlista?

200 DATA "ALMA", KÖRTE, "SZILVA"

5. Keressük ki, hogy az előző fejezet feladatai közül melyiket lehet megoldani a READ és a DATA utasításokkal!

6. Írjunk programot

a) amivel karakterekből kialakított betűkkel szavakat írhatunk ki;
b) amivel a cirill betűket jeleníthetjük meg;
c) vasúti jegyek árának meghatározásához;
d) bérek címletezéséhez!

CIKLUS

A ciklusszervezéssel kapcsolatos legfontosabb fogalmakkal már megismerkedtünk. Fontosságukra való tekintettel rendszerezzük azokat:

Ciklus: programrészlet és annak ismételt végrehajtását biztosító utasítások
Ciklusmag: az a programrészlet, aminek végrehajtása ismétlődik.
Vizsgálat: annak eldöntése, hogy az ismétlődés folytatódjon-e.
Ciklusváltozó: az a változó, amelyiknek az értékétől függ a program futása során a ciklus befejezése.
Előkészítés: azok az utasítások, amelyekkel a ciklusbalépés előtt a változók kezdőértékeit megadjuk.

A *ciklusszervezés* a ciklus megtervezését jelenti. Mit kell végiggondolnunk, ha a programban ciklust akarunk létrehozni?

Mi ismétlődik?

Akkor, amikor egy műveletsort többször végre akarunk hajtatni a számítógéppel, gondoljunk arra, hogy ciklust szervezzünk. Az ismétlődő programrészletet fogalmazzuk meg általánosan, változók segítségével. Ez lesz a *ciklusmag*.

Mi változik?

A ciklusmag végrehajtása során legalább egy változó legyen, aminek az értéke változik. Amennyiben ilyen nincs, nekünk kell gondoskodnunk a *ciklusváltozóról*. A ciklus-változó értékétől függ, hogy mikor fejeződik be a ciklus.

Mit vizsgálunk?

Az ismétlődést valamilyen *feltételtől* tesszük függővé. Ez lehet egy vagy több változó értékének függvénye. Azok a változók, amelyeknek értékei szerepet játszanának annak eldöntésében, hogy befejeződik-e a ciklus, azokat *ciklusváltozókként* kezeljük.

Milyen kezdőértékei lesznek a változóknak?

Mielőtt belépnénk a ciklusba, *értéket kell adnunk* a ciklusmagban található változóknak.

Amennyiben nem adunk a változóknak értéket, a ciklusba történő belépés előtt, a numerikus változók értéke automatikusan 0, a karakterlánc-változóké „üres karakter".

● Pozitív számokból álló számhalmazból válasszuk ki a legkisebbet!

Az adatokat a DATA adatlistából egymás után be kell olvasni, és mindegyiket össze kell hasonlítani a korábban beolvasott adatok közül kiválasztott legkisebbel. *Ez ismétlődik.*

Az adatsor végét kell felismernie a programnak, hogy *a ciklus befejeződjön.* Az előző fejezetben láttuk, hogy erre két lehetőségünk van:

a) A beolvasott adatok változnak. Használjuk ezt ki. Írjunk olyan adatot az adatsor végére, ami biztosan nem szerepel az adatok között. Feladatunkban lehet az a feltétel, hogy akkor fejeződjön be a ciklus, ha a beolvasott érték 0.

Hogyan lépünk a ciklusba? Mihez hasonlítjuk az első elemet? Az M változóba helyezzük el mindig a kisebb értéket. Legyen az M *kezdőértéke* egy olyan nagy szám, aminél biztos, hogy a számhalmaz elemei kisebbek.

```
10  REM LEGKISEBB-1
20  M=1E38
30  READ X
40  IF X=0 THEN 70
50  IF X<M THEN M=X
60  GOTO 30
70  PRINT "A LEGKISEBB";M
80  DATA 123,47,58,3,9
90  DATA 0
```

b) Akkor, ha ismerjük az adatok számát, a *ciklusváltozóról* nekünk kell gondoskodnunk. Legyen ez az I változó. Az I változó értékét egyesével növeljük a ciklusmagon belül. Ez a beolvasott adatok számlálását jelenti. A vizsgálatot I változó értékével végezzük.

Az előző megoldásban észrevehetjük, hogy az M értéke már az első adat beolvasása után megváltozik. Legyen most az M kezdőértéke az első adat.

```
10  REM LEGKISEBB-2
20  INPUT "AZ ADATOK SZAMA",N
30  READ M
40  I=2
50  READ X
60  IF X<M THEN M=X
70  I=I+1
80  IF I<=N THEN 50
90  PRINT "A LEGKISEBB",M
100 DATA 123,47,58,3,9
```

Vegyük észre, hogy a két program között a leglényegesebb különbség az, hogy az elsőnél nem tudjuk előre megmondani, hogy a ciklusváltozó hogyan változik, a másodiknál a változás ismert. Ismerjük a ciklusváltozó kezdő és végértékét is.

A második programban szereplő ciklushoz hasonló ciklusok létrehozására szolgál

a FOR/NEXT utasításpár, amit *ciklusképző utasításpárnak* neveznek. Ezzel az utasítással a feladat megoldása:

```
 10  REM LEGKISEBB-3
 20  INPUT "AZ ADATOK SZAMA",N
 30  READ M
 40  FOR I=2 TO N
 50  READ X
 60  IF X<M THEN M=X
 70  NEXT I
 80  PRINT "A LEGKISEBB",M
100  DATA 123,47,58,3,9
```

A FOR/NEXT utasításpár gondoskodik a ciklusváltozó kezdőértékének megadásáról, a ciklusváltozó értékének változtatásáról és a vizsgálatról is. A FOR és a NEXT utasítások között kell elhelyeznünk a ciklusmagot.

A FOR és a NEXT utasítások mindig csak párban szerepelhetnek a programban. Az összetartozást azzal jelezzük, hogy a NEXT utasításban is szerepel az a ciklusváltozó, amelyikre vonatkozik.

A ciklusváltozó értéke a programunkban 1-gyel nő minden ismétlésnél mindaddig, ameddig nagyobb nem lesz, mint a végérték. A programunkban a ciklusváltozó kezdőértéke 2, a végértéke N.

A FOR utasítást megadhatjuk a következő alakban is:

```
40  FOR I=2 TO N STEP 1
```

Az utasítás teljes alakját csak akkor kell használni, ha a lépésköz 1-től különböző.

A ciklusképző utasítás:

FOR ciklusváltozó = kezdőérték TO végérték STEP lépésköz.
—
—
— a ciklusmag utasításai
—
—
NEXT ciklusváltozó

A ciklusváltozó kezdő és végértékét, valamint a lépésközt megadhatjuk tetszőleges aritmetikai kifejezéssel, de az nem lehet kétszeres pontosságú.

Az utasítás leírása:

1. Vegye fel a ciklusváltozó a kezdőértéket!
2. Fusson az azonos ciklusváltozójú NEXT utasításig a program!
3. Növelje a ciklusváltozó értékét az adott lépésközzel!
4. Vizsgálja meg, hogy a ciklusváltozó a kezdő és a végérték közé esik-e, és ha igen:
   2. ponton folytassa!
   ha nem: folytatódjon a program végrehajtása a NEXT utáni utasítással!

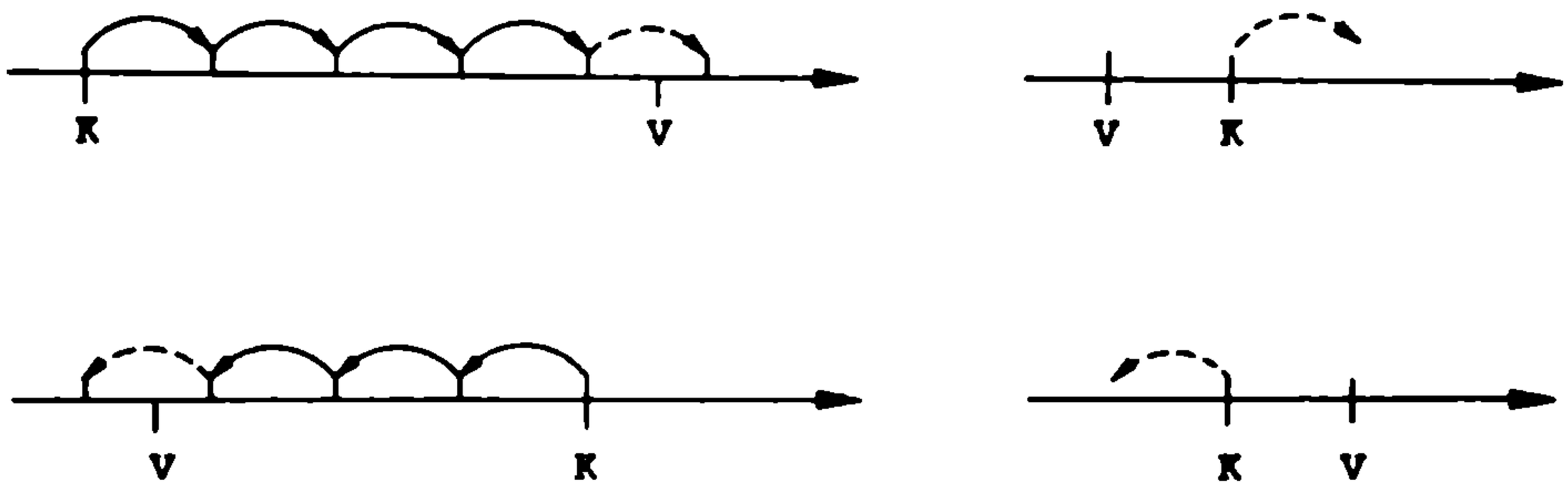
A következő kis programmal kitapasztalhatjuk a FOR/NEXT utasítás működését.

```
10  INPUT "KEZDOERTEK",K
20  INPUT "VEGERTEK",V
```

```
30  INPUT "LEPESKOZ",L
40  FOR X=K TO V STEP L
50  PRINT "CIKLUSMAG",X
60  NEXT X
70  PRINT "KILEPTUNK A CIKLUSBOL",X
```

Akkor, ha a lépésköz pozitív, a ciklusváltozó értéke növekszik, ha negatív, csökken.

Ábrázoljuk a számegyenesen a kezdő és végértékeket, valamint a ciklusváltozó értékeit (25. ábra). A kezdő értékkel minden esetben végrehajtódik a ciklus. A módosított értékű ciklusváltozóval csak akkor, ha az értéke a kezdő és a végérték közé esik vagy a végértékkel egyenlő. Szemléletesen ezt úgy értelmezhetjük, hogy ha a ciklusváltozó értékeivel a végérték felé haladunk, és ha azon még nem léptünk túl, akkor a ciklusmag végrehajtódik.



25. ábra

A 25. ábrán szaggatott vonallal jeleztük annak az értéknek a felvételét, amivel már kilépünk a ciklusból. A 70-es sorban ezt az értéket íratjuk ki.

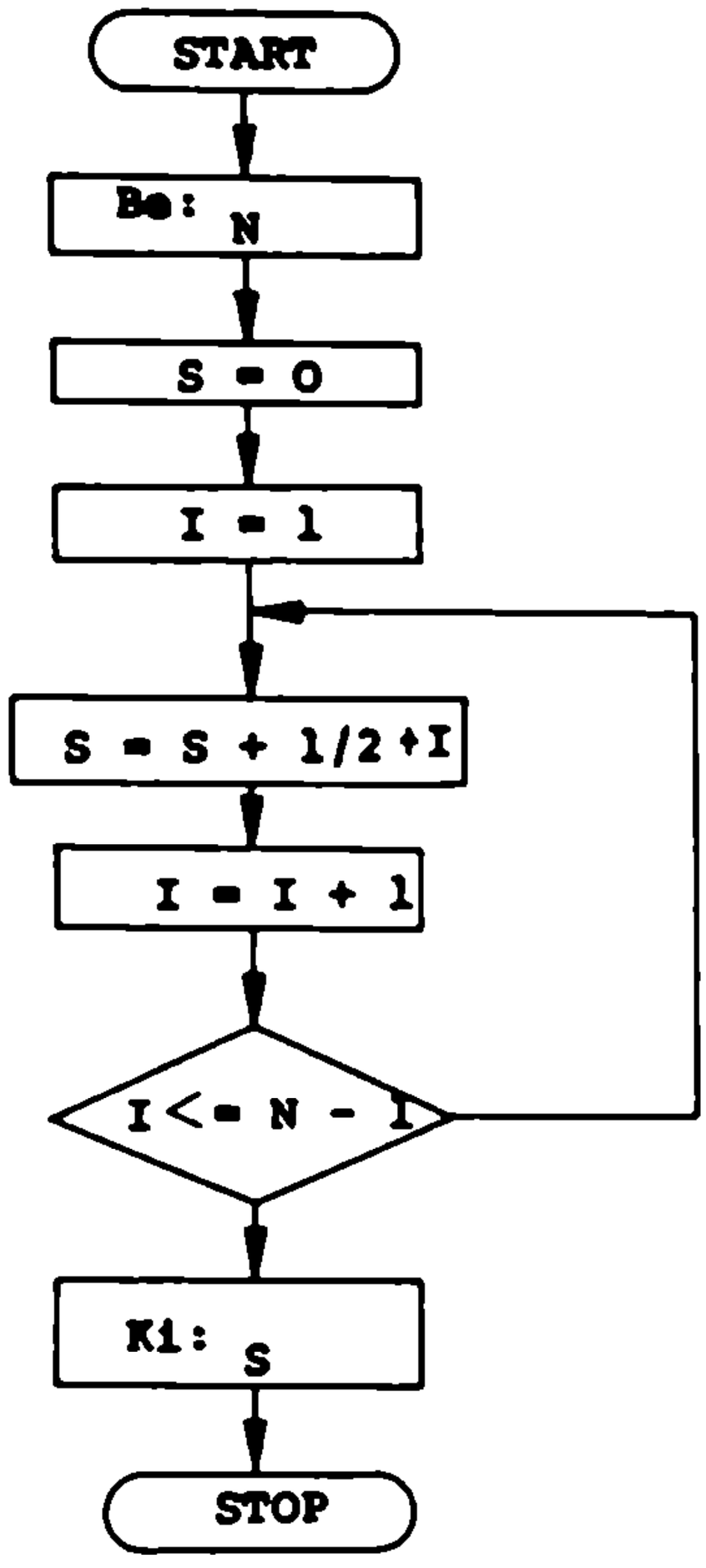
● Képezzük az $a_i = \frac{1}{2^i}$ sorozat első n elemének összegét, ahol $i = 0, 1, 2 \ldots n!$

Ciklusszervezéskor arra ügyeljünk ebben a feladatban, hogy az első szám kitevője 0, ezért a ciklusváltozóval vagy számlálunk, vagy a kitevőket jelöljük. Az utóbbi esetben az n-edik elem kitevője $n-1$.

```
10  REM SOROZAT ELEMEINEK OSSZEGE-1
20  S=0:INPUT "ELEMEK SZAMA",N
30  FOR I=0 TO N-1
40  S=S+1/2 ↑ I
50  NEXT I
60  PRINT N, "ELEM OSSZEGE",S
```

A ciklusutasítás elöször változtatja a ciklusváltozó értékét, és csak ezután történik a vizsgálat. A következő program jobban szemlélteti az ilyen szerkezetű ciklust (26. ábra).

```
10   REM SOROZAT ELEMEINEK OSSZEGE-2
20   S=0:INPUT "ELEMEK SZAMA",N
30   I=0
40   S=S+1/2 ↑ I
50   I=I+1
60   IF I <=N-1 THEN 40
70   PRINT N, "ELEM OSSZEGE",S
```



26. ábra

Utólagos vizsgálattal szervezett ciklus

Az utólagos vizsgálattal szervezett ciklusok általános sémáját mutatjuk be a 27. ábrán.

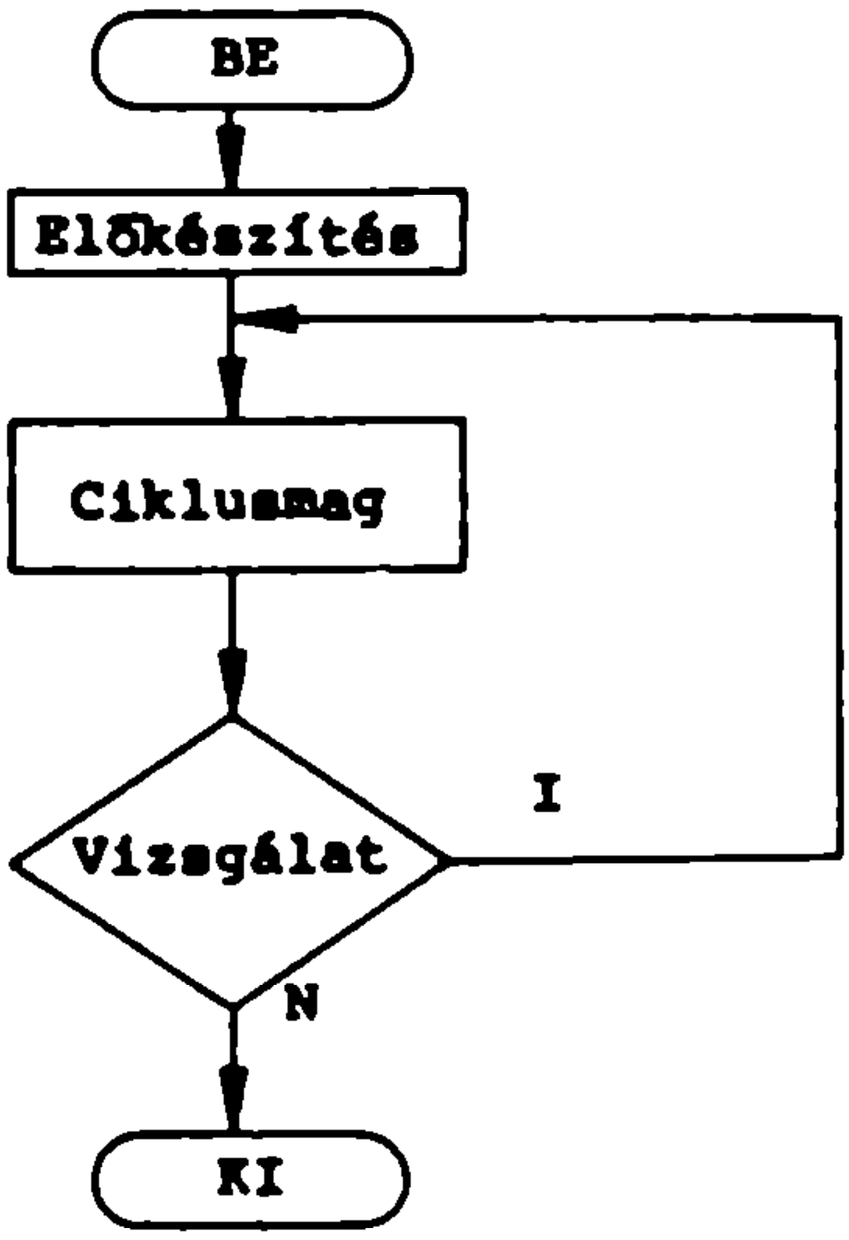
• Összegezzük az $a_i = \frac{1}{2^i}$ sorozat elemeit mindaddig, ameddig a_i nagyobb mint 1E–15 (i = 0; 1; 2. . .)!

A sorozat elemeit osztással állítjuk elõ, az elõzõ elem ismeretében:

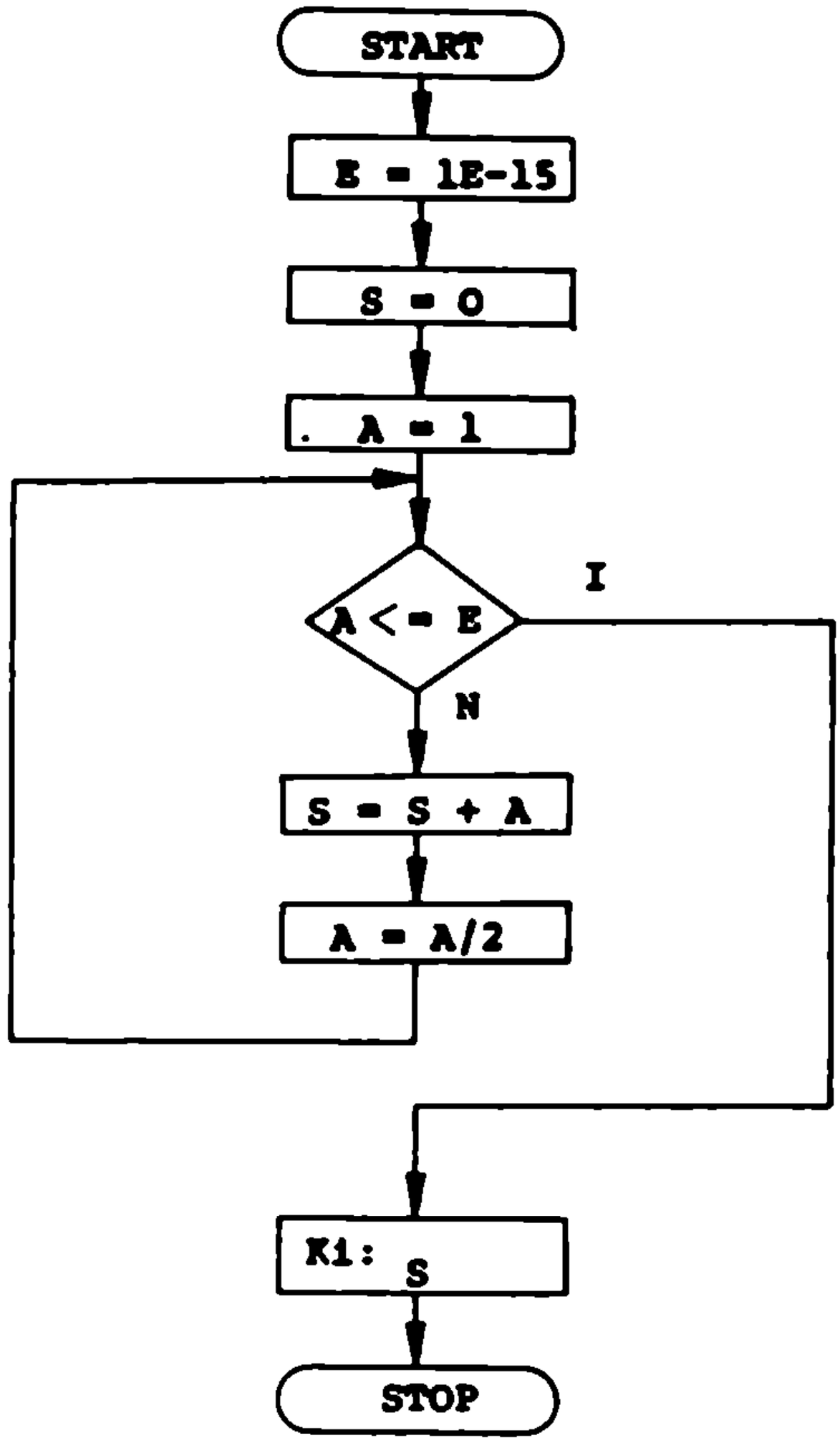
$$a_i = \frac{a_{i-1}}{2} .$$

A programunk ezzel gyorsabb lesz, és kétszeres pontossággal számolhatunk, mivel nem használjuk a hatványozást.

Nem tudjuk előre, hogy hányadik elemnél lépjük át a határt, ezért az összegezés előtt kell a vizsgálatot elvégeznünk (28. ábra):



27. ábra



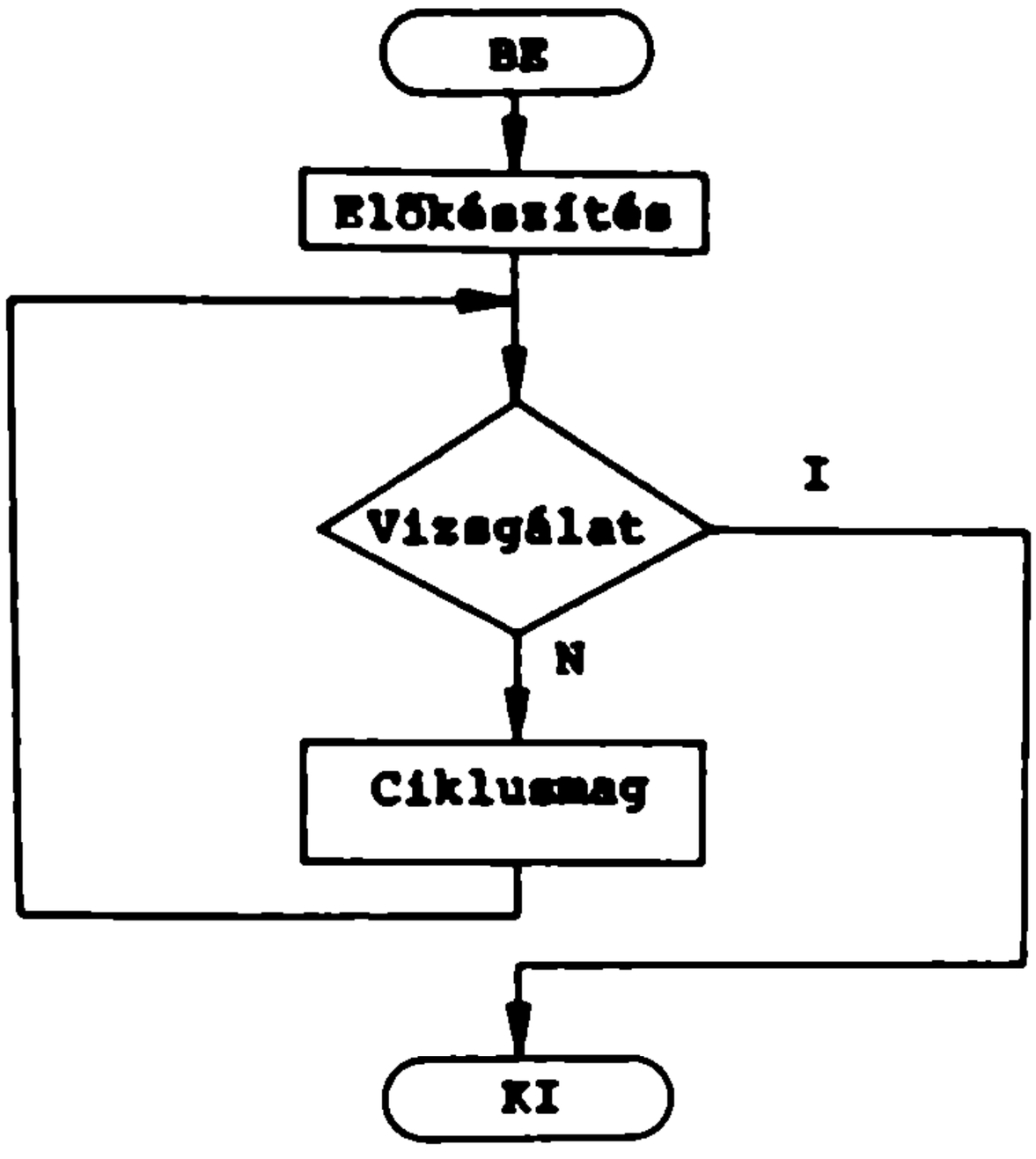
28. ábra

```
10   REM SOROZAT ELEMEINEK OSSZEGE-3
20   DEFDBLA-Z:CLS
30   E=1E-15
40   S=0:A=1
50   IF A<=E THEN 90
60   S=S+A:PRINT@0,S,A
70   A=A/2
80   GOTO 50
90   PRINT "OSSZEG",S
```

A 60-as sorban a PRINT@0, S, A utasítással kiíratjuk a részösszegeket és az elemeket, így a közelítést követni tudjuk.

Előzetes vizsgálattal szervezett ciklus

Az *előzetes vizsgdlattal* szervezett ciklusok általános sémáját a 29. ábra mutatja.



29. ábra

A változók értékeinek változását figyelemmel tudjuk kísérni, ha beírjuk a következő utasítássort:

```
75 IF INKEY$ =" " THEN 75
```

A program mindaddig várakozik ennél az utasításnál, ameddig le nem nyomunk egy billentyűt.

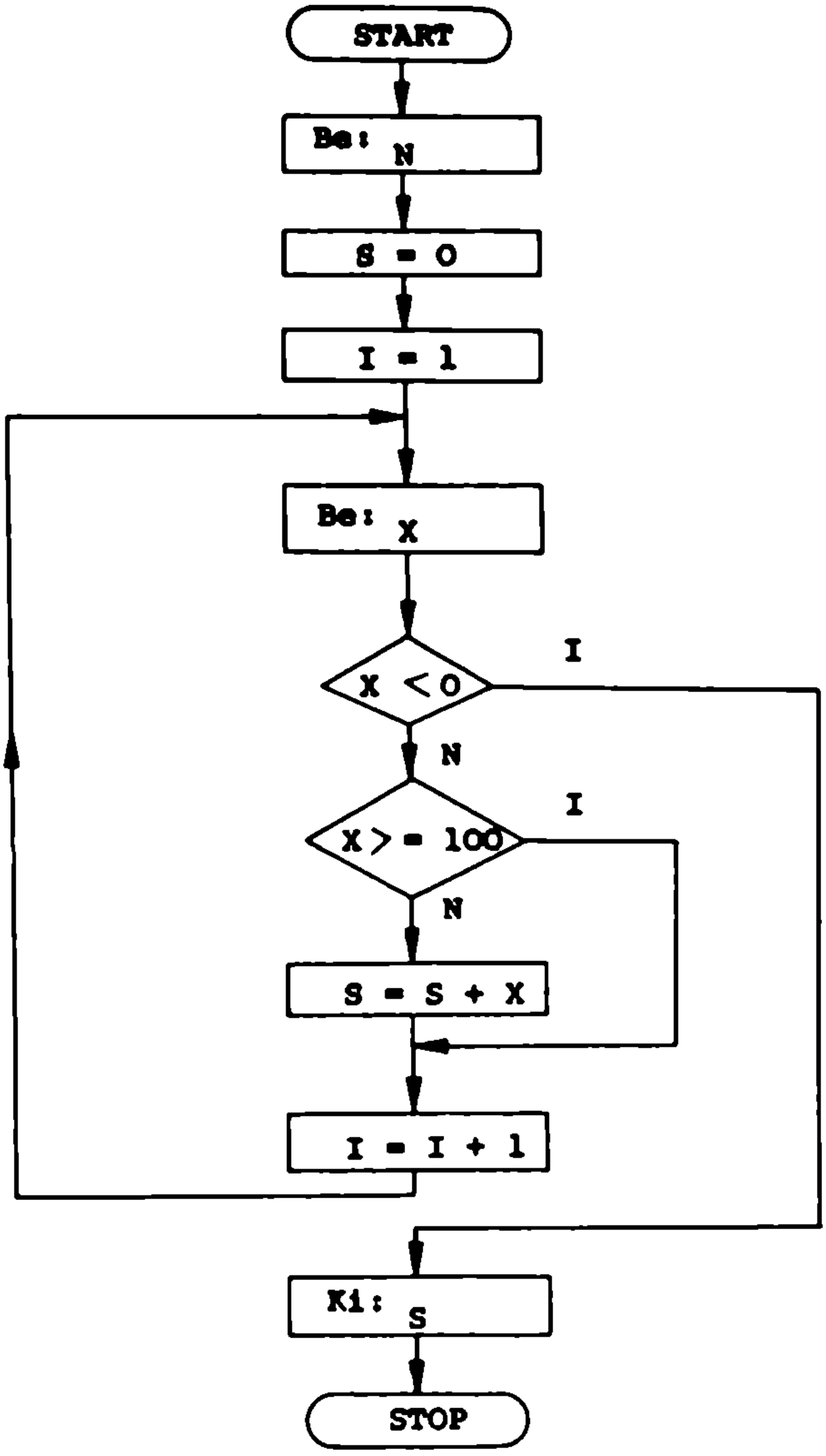
● Összegezzük egy n elemű számhalmaz 100-nál kisebb elemeit az első negatív számig!

A feltételek szerint meg kell vizsgálnunk a beolvasott elemeket, és csak a 100-nál kisebb pozitív számokat kell összegeznünk. Negatív szám esetén ki kell ugranunk a ciklusból.

A program folyamatábrája (30. ábra):

A feladat szerint N számot kell összegeznünk, tehát az N elem beolvasásához a FOR/NEXT ciklusképző utasítást használhatjuk. Akkor, ha negatív számot találunk a beolvasott adatok között, korábban kilépünk a ciklusból.

A 100-nál kisebb számok kiválasztása a 70-es sorral történik. Akkor, ha nem összegezzük a beolvasott számot, a NEXT utasításra ugrunk. A ciklusmag befejezése, a ciklusváltozó egy adott értékénél csak így történhet, a NEXT utasítás végrehajtásával.



30. ábra

```
 10  REM KIKERESES OSSZEGZES
 20  INPUT "ELEMEK SZAMA",N
 30  S=0
 40  FOR I=1 TO N
 50  READ X
 60  IF X<0 THEN 100
 70  IF X>100 THEN 90
 80  S=S+X
 90  NEXT I
100  PRINT "AZ OSSZEG",S
110  DATA 123,41,2,3,56,0,36,
120  DATA 77,-13,56,47,-46,32
```

● Képezzük az A×B direkt szorzatot!

$A = \{ a, a+1, \ldots, b-1, b \}$
$B = \{ c, c+1, \ldots, d-1, d \}$

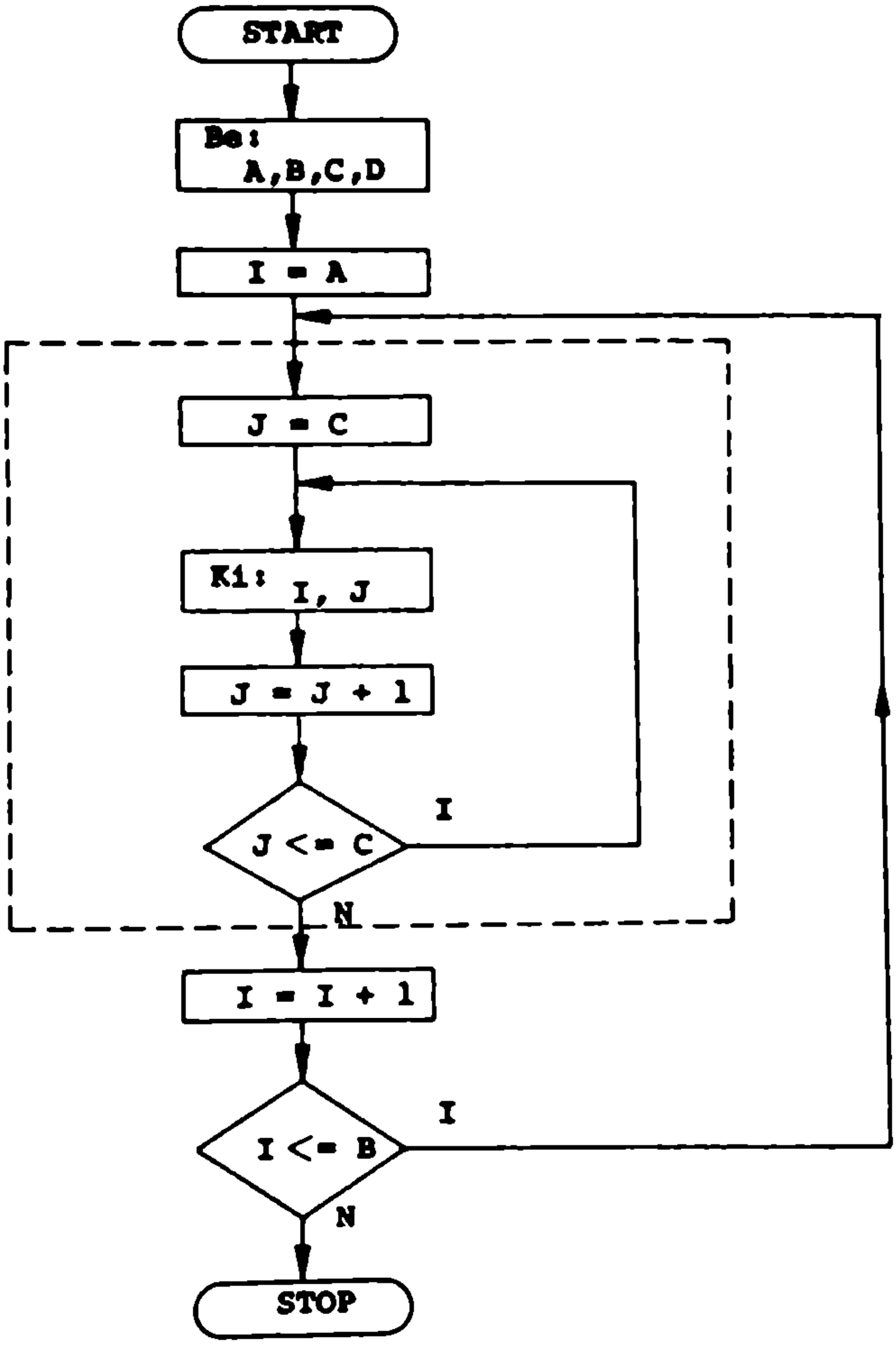
ahol a, b, c és d egész számok.

A feladat az, hogy A halmaz minden elemét párosítani kell a halmaz elemeivel.

Válasszuk ki A egy elemét, és állítsuk párba B minden elemével. B elemeinek előállítása, mivel azok egymást követő egész számok, egy ciklussal történhet.

Az A halmaz összes elemét ugyancsak egy ciklussal tudjuk megadni, amiket azután felhasznál a másik ciklus.

A két ciklust egymásba kell építeni (31. ábra).



31. ábra

A szaggatott vonallal körülhatárolt rész lesz a *belső ciklus*, amit a *külső ciklus* ciklusmagjában helyeztünk el.

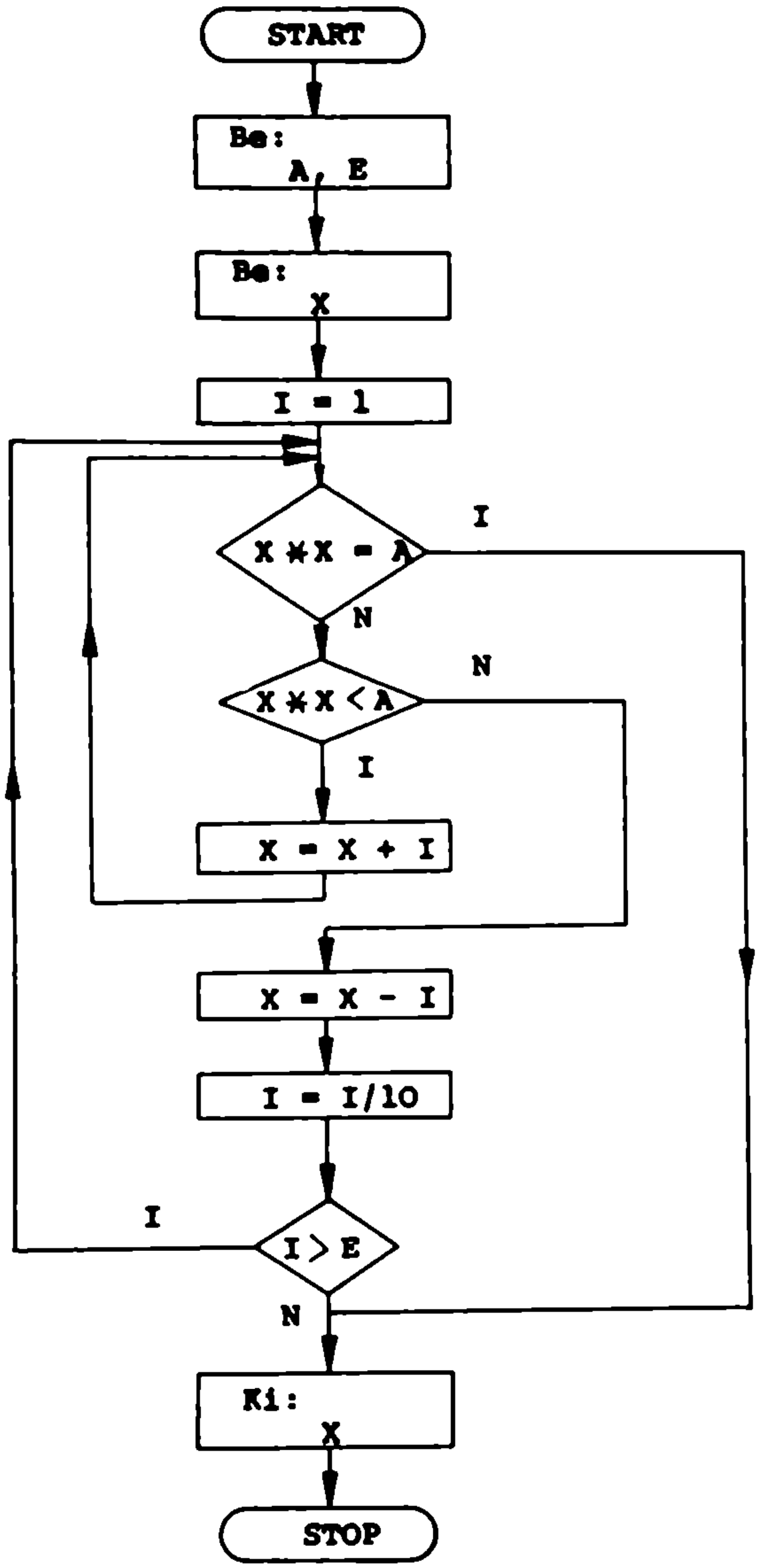
Fontos követelmény, hogy a FOR/NEXT utasításpárokat összekötő vonalak *nem*

keresztezhetik egymást. Ez azt jelenti, hogy az egyik ciklus magában kell hogy foglalja a másik ciklust.

● Közelítsük meg egy szám négyzetgyökét egész, tized, század stb. lépésközönként „alulról"!

Az „alulról" közelítés azt jelenti, hogy addig növeljük egy változó értékét, amíg négyzete nagyobb nem lesz, mint az a szám, amelyiknek a négyzetgyökét keressük.

A közelítés léptéke mindig változik. Az előző lépték tizedrésze lesz. A léptéket a külső ciklus ciklusváltozójával adjuk meg. Ez az I változó értéke lesz. 32. ábra a program folyamatábrája.



32. ábra

```
 10  REM NEGYZETGYOK KOZELITES ALULROL
 20  DEFDBL A-Z
 30  CLS:INPUT "A SZAM",A
 40  INPUT "PONTOSSAG",E
 50  INPUT "A SZAM NEGYZETGYOKENEL KISEBB
     EGESZ SZAM",X
 60  I=1:REM A KOZELITES LEPTEKE
 70  IF X*X=A THEN 120
 80  IF X*X<A THEN X=X+I:PRINT X:GOTO 70
 90  X=X-I
100  I=I/10
110  IF I>E THEN 70
120  PRINT " GYOK",X
```

A 70-es sorban vizsgáljuk az egyenlőséget. Ez a program kritikus pontja. Négyzetszámok esetén nem kapunk helyes értéket. A számítógép pontatlanul számol, mert a tízes számrendszerbeli számokat kettes számrendszerben ábrázolja a memóriában. Ez az átalakítás és visszaalakítás hibaforrásként jelentkezik. Ezt figyelembe kell vennünk.

Az egyenlőség vizsgálata helyett azt vizsgáljuk, hogy a közelítő érték nem esik-e egy adott intervallumba. Az intervallum szélességét a pontossággal adjuk meg.

```
70  IF X*X>A-E AND X*X<A+E THEN 120
```

- Határozzuk meg két szám legnagyobb közös osztóját!

Keresnünk kell azt a legnagyobb számot, amelyik mindkét számnak osztója. Legyen a két szám a és b, a legnagyobb közös osztó k, akkor felírható, hogy

$$a = k \cdot a' \quad \text{és} \quad b = k \cdot b'.$$

k osztója lesz a két szám különbségének is: $a-b = k \cdot a' - k \cdot b' = k(a'-b')$, ami biztosan kisebb, mint a két szám közül a nagyobb.

A nagyobb szám helyett mindig vesszük a két szám különbségét, és ezt ismételjük mindaddig, míg a különbség nem nulla, azaz a két szám nem egyenlő. Egyenlőség esetén a számok a legnagyobb közös osztót adják!

A két vizsgált szám értékének a csökkenése miatt biztos, hogy véges lépésben meghatározhatjuk az eredeti számok legnagyobb közös osztóját.

Az algoritmusunk lépésenként leírva:

1, Adott két szám.
2. A két szám egyenlő?
   igen: 5. ponttal folytasd!
   nem: 3. ponttal folytasd!
3. A nagyobb szám helyett vegyük a két szám különbségét!
4. 2. ponttal folytasd!
5. Ez lesz a legnagyobb közös osztó!

Használjuk fel az algoritmus leírásához a programnyelv jelöléseit!

1. A két szám: A, B.
2. A = B?

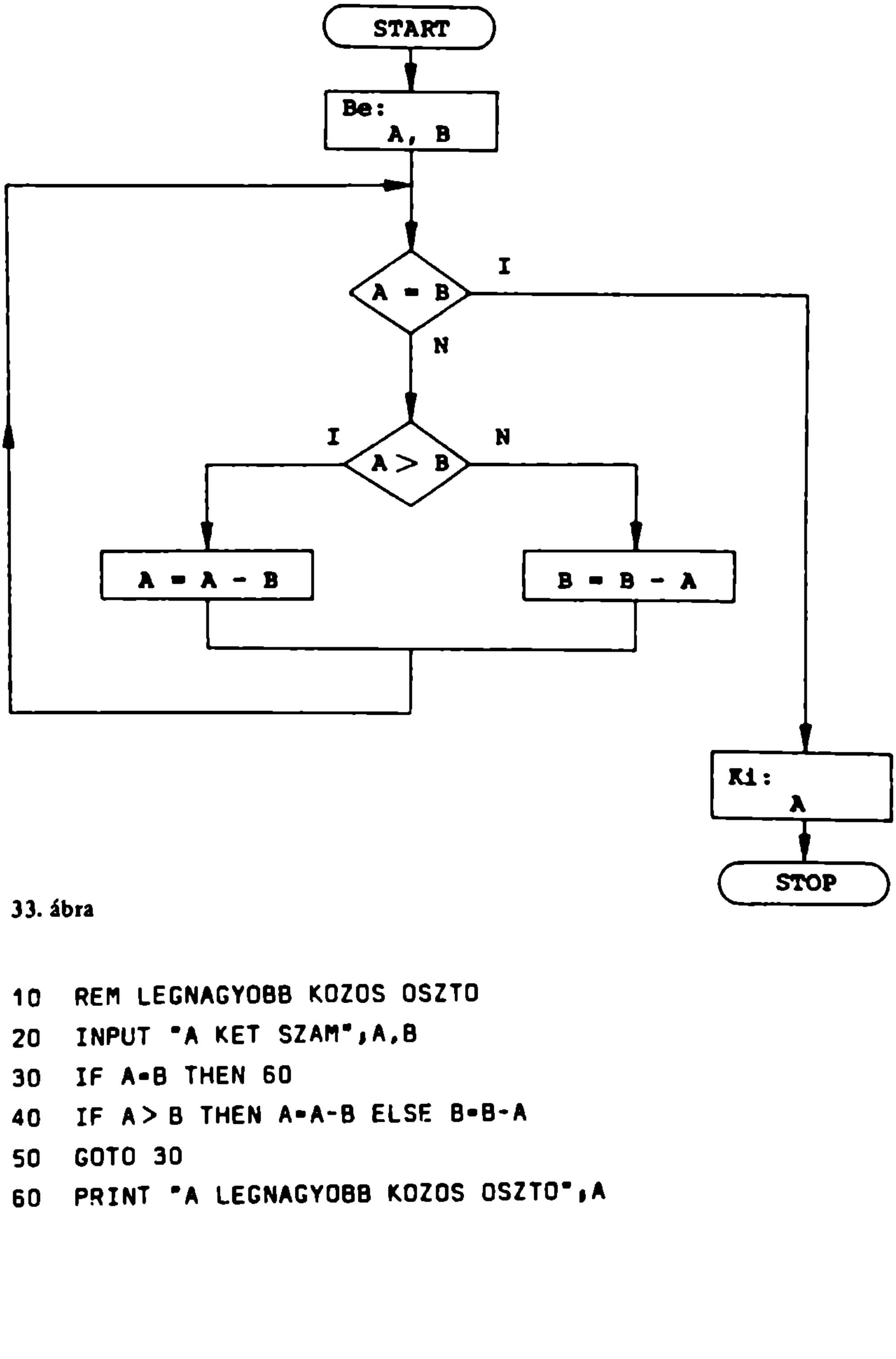
   Igen: 5. ponttal folytasd!

   igen: A = A---B

   nem: B = B - A.

3. A > B?
4. 2. ponttal folytasd!
5. A legnagyobb közös osztó A!

A két szám legnagyobb közös osztóját meghatározó algoritmus folyamatábrája (33. ábra):



33. ábra

```
10  REM LEGNAGYOBB KOZOS OSZTO
20  INPUT "A KET SZAM",A,B
30  IF A=B THEN 60
40  IF A>B THEN A=A-B ELSE B=B-A
50  GOTO 30
60  PRINT "A LEGNAGYOBB KOZOS OSZTO",A
```

Kérdések és feladatok

1. Ismertessük a ciklusszervezéssel kapcsolatos fogalmakat!
2. Ismertessük a ciklus szerkezetét! Ábrázoljuk folyamatábrával!
3. Írjunk programot, amelyikkel az INPUT utasítással beolvasott adatok közül
   a) a legkisebbet,
   b) a legnagyobbat,
   c) a legkisebbet és a legnagyobbat is kiválasztjuk!
4. Milyen ciklus létrehozásakor alkalmazhatjuk a FOR/NEXT ciklusképző utasítást?

Keressük ki az előző mintaprogramok közül ezeket, és oldjuk meg a feladatokat a FOR/NEXT utasításpárral is!

5. Készítsünk folyamatábrát a ciklusképző utasítás működésének leírásához!
6. Írjunk programot, amivel π közelítő értékét kiszámíthatjuk!

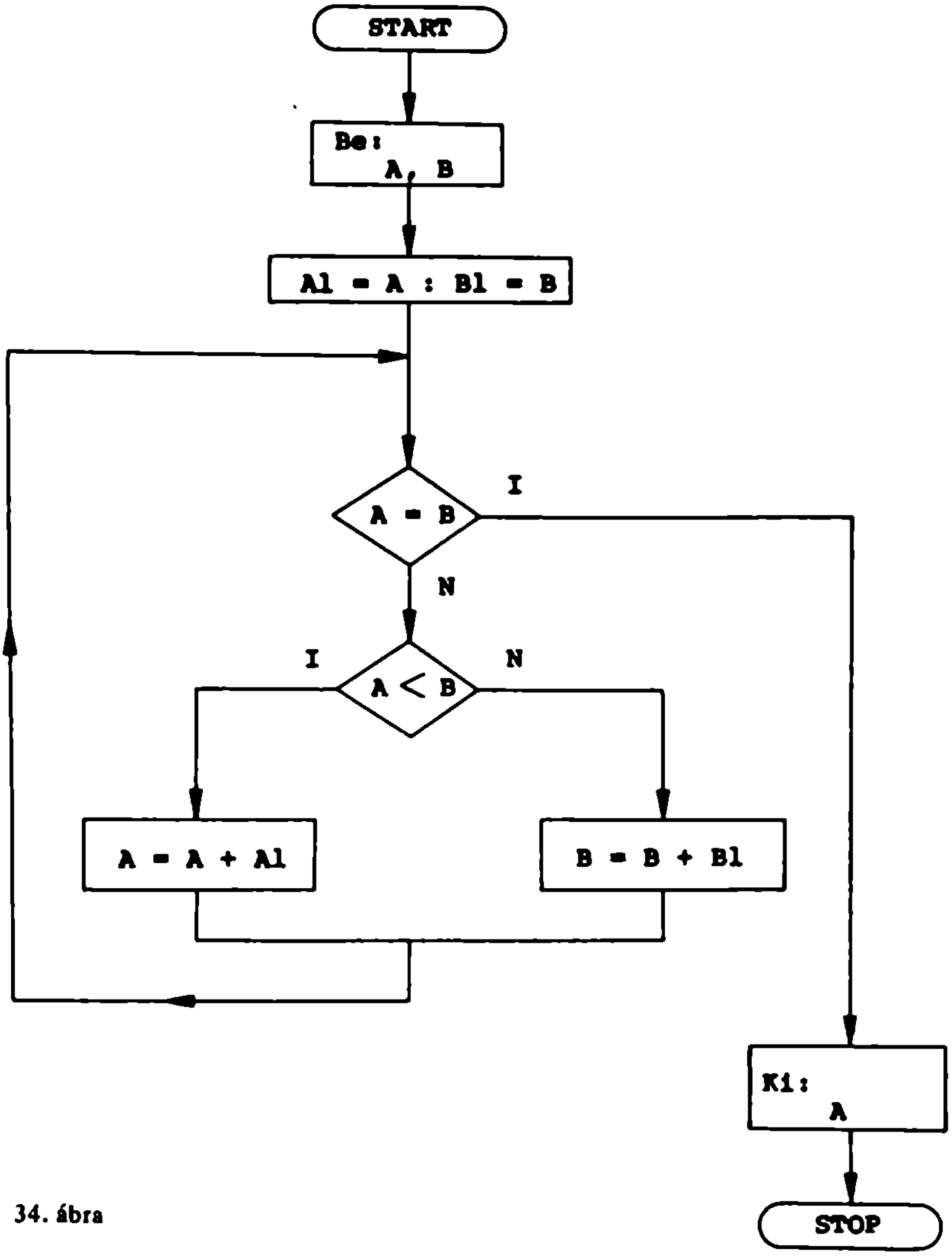
$$\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdots$$

(Wallis, 1659)

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \cdots$$

(Leibnitz, 1673)

7. Hogyan fejeződik be a ciklusmag a FOR/NEXT utasításban?
8. Írjunk programot, amivel meghatározhatjuk egy numerikus adathalmaz 3-mal osztható páros számait!
9. A 34. ábra két szám legkisebb közös többszörösét előállító algoritmus folyamatábrája! Írjuk meg a programot!



34. ábra

10. Hogyan határozható meg két szám legkisebb közös többszöröse, ha a két szám legnagyobb közös osztóját ismerjük?

11. Hogyan oldjuk meg a kettes számrendszerben adott számok tízes számrendszerbe történő átírását ciklus segítségével?

12. Írjunk programot, amivel előállíthatjuk egy szó betűiből a szó morze kódját! A betűket és azok kódjait a DATA adatlistában helyezzük el a következő formában:

```
300  DATA .-,A,-...,B,-.-.,C
```

13. Mire kell ügyelni két ciklus egymásba ágyazásakor?

14. Készítsünk három számjegyű „kilométerórát", ami a nyolcas (oktális) számrendszerben számlál!

15. Írjunk programot, amivel függvények értéktáblázata készíthető!

16. Értelmezzük a következő programot, és készítsük el a folyamatábráját!

Próbáljuk meg gyorsítani a programot!

```
10  CLS:INPUT A
20  B=A-2:DEFINT I
30  I=A/B
40  IF I=A/B THEN PRINT B
50  B=B-1:IF B>1 THEN 30
```

ARITMETIKAI FÜGGVÉNYEK

Az aritmetikai kifejezésekben közvetlenül hivatkozhatunk az 5. mellékletben megadott elemi függvényekre. Ezeket a függvényeket, amint azt látni fogjuk, nemcsak függvényeket tartalmazó kifejezések értékének meghatározására használhatjuk, hanem programozási feladatok megoldásában is segítségünkre lehetnek. A függvények gyakorlati alkalmazására látunk példákat programjainkban.

A *függvények* három v. négybetűs azonosító neve a matematikai jelölésükkel azonos vagy arra utal.

A függvény független változójának értékét a függvény neve után a zárójelben megadott kifejezés értéke adja. Ez a függvény *argumentuma.*

```
SQR(A*A+B*B)     ABS(Y-(A*X+B))
SIN(A*3.1415/180)    EXP(LOG(X)/N)
```

Az argumentumnak a függvény értelmezési tartományába kell esni. Az *értelmezési tartományt* a függvény matematikai értelmezése és az argumentumban szereplő kifejezés lehetséges pontossága, annak számtartománya együtt határozza meg.

Az SQR(X) függvény negatív számokra nincs értelmezve, és az argumentum egyszeres pontosságú lehet, ezért az értelmezési tartomány: $0 \leqslant X \leqslant 1.701411E38$.

● Adott három szakasz. Határozzuk meg a szakaszokból szerkeszthető háromszög kerületét és területét!

Mielőtt a számításokhoz hozzákezdenénk, a háromszögek oldalaira vonatkozó egyenlőtlenségek felhasználásával megvizsgáljuk, hogy a három szakasz háromszöget alkot-e. Ha igen, kiszámítjuk kerületét és a Heron-képlet alkalmazásával a területét.

$$K = a+b+c \qquad t = \sqrt{s(s-a)(s-b)(s-c)}$$

ahol $s = \frac{k}{2}$.

```
10  REM HAROMSZOG KERULETE ES TERULETE
20  CLS:INPUT "OLDALAK",A,B,C
30  IF A+B<=C OR A+C<=B OR B+C<=A THEN "NEM
    LEHET HAROMSZOG":END
40  K=A+B+C:S=K/2
50  T=SQR(S*(S-A)*(S-B)*(S-C))
60  PRINT "KERULETE",K
70  PRINT "TERULETE",T
```

A négyzetgyök függvény helyett alkalmazhatjuk a törtkitevőjű hatványozást is:

```
50  T=(S*(S-A)*(S-B)*(S-C))↑0.5
```

Páros gyökkitevő esetén a négyzetgyök függvényeket egymásba építhetjük, ami azt jelenti, hogy a függvény argumentumát meghatározó kifejezés is tartalmazhat függvényt.

X változó értékének negyedik gyöke:

```
SQR(SQR(X))
```

Ehelyett célszerűbb itt is a törtkitevő használata:

```
X ↑ 0.25
```

Páratlan kitevőjű gyökök kiszámítását is a törtkitevőjű hatványozással oldhatjuk meg: $\sqrt[5]{X}$ kiszámítása a programban: X ↑ (1/5).

A logaritmus azonosságait felhasználva, a LOG és EXP függvények segítségével is meghatározhatjuk a gyököket.

● Írjunk programot, amivel kétszeres pontossággal kiszámíthatjuk egy szám négyzetgyökét!

Alkalmazzuk a Newton-féle iterációt. Lépésenként közelítjük a szám négyzetgyökét.

Az eljárás lényege, hogy ha $x_i < \sqrt{x}$, akkor $\frac{x}{x_i} > \sqrt{x}$, és ekkor x_i és $\frac{x}{x_i}$ számtani közepe

$$x_{i+1} = 0{,}5 \left[x_i + \frac{x}{x_i} \right]$$

x négyzetgyökének egy pontosabb megközelítését adja. Ugyanez érvényes, ha $x_i > \sqrt{x}$, mert akkor $\frac{x}{x_i} < \sqrt{x}$. x_i az iteráció i. képzésében a közelítő érték.

Az első közelítő érték $x_1 = \frac{x}{2}$.

Az i. közelítő érték ismeretében meghatározzuk az i+1. közelítő értéket, ami a következő lépésben az i. szerepét tölti be.

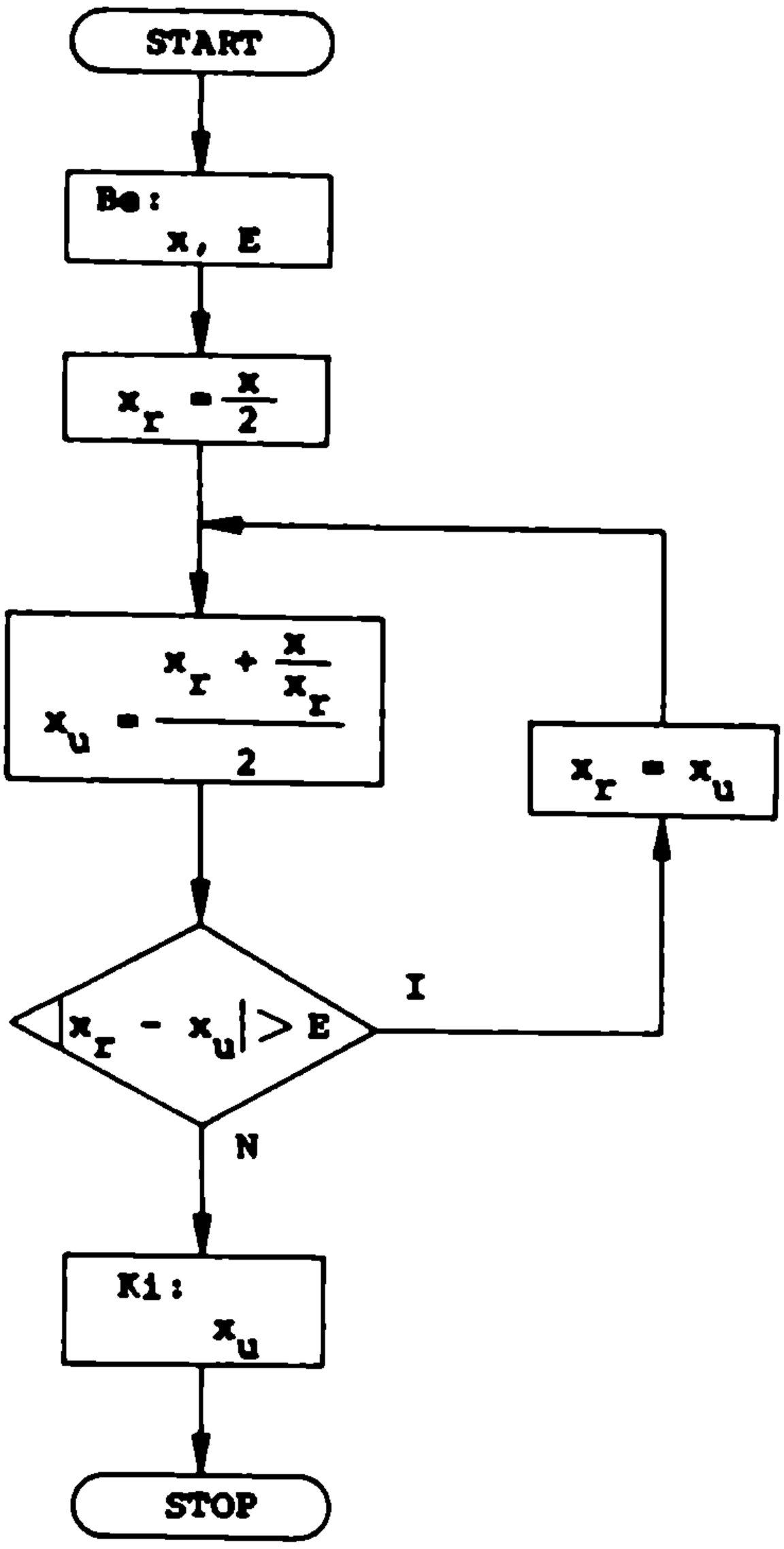
Az i. közelítő értéket nevezzük „régi", az i+1.-et „új" közelítő értéknek. Ez jobban kifejezi azt, hogy az iterációval egy olyan értéket állítunk elő, amit a következő lépésben már mint kiindulási értéket veszünk figyelembe.

Az E értéke jelentse a közelítés pontosságát. Amikor az „új" és a „régi" közelítő érték közelebb van egymáshoz, mint E ($E > 0$), akkor az iterálást befejezzük (35. ábra).

A változókat kétszeres pontosságúaknak deklaráltuk a DEFDBL utasítással.

A függvények között van olyan, aminek az argumentuma kétszeres pontosságú érték is lehet. Ilyen az *abszolútérték függvény* is, amivel az XR és az XU változó értékével adott intervallum hosszát határozzuk meg. Az abszolútérték függvény függvényértéke is kétszeres pontosságú lesz.

Ezzel a programmal bemutattuk azt is, hogy a függvényértékek kiszámítására vannak olyan matematikai eljárások, amelyekben csak a négy alapműveletet használjuk.



35. ábra

```
10   REM NEWTON-FELE ITERACIO
20   DEFDBL X,E:CLS
30   INPUT "A SZAM",X
40   INPUT "PONTOSSAG",E
50   XR=X/2
60   XU=.5*(XR+X/XR)
70    IF ABS(XU-XR)>E THEN XR=XU: GOTO60
80  PRINT X,"NEGYZETGYOKE",XU
```

● Adott két pozitív szám. Osszuk el maradékosan a nagyobb számot a kisebbel!

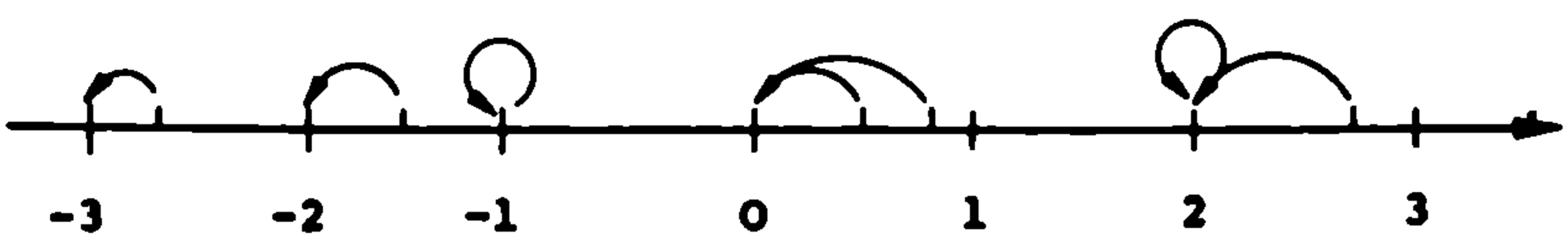
A FIX(X) *egészrész függvény* értéke az az egész szám lesz, amit úgy kapunk, hogy elhagyjuk X értékének tizedes részét. „Levágjuk" a tizedes számjegyeket.

A két pozitív szám hányadosának egész része tehát megadja, hogy hányszor van meg a nagyobb számban a kisebb. A maradékot ebből már meg tudjuk határozni!

```
10  REM MARADEKOS OSZTAS
20  INPUT "A KET SZAM",A,B
30  IF A<B THEN T=A:A=B:B=T
40  H=FIX(A/B):M=A-B*H
50  PRINT A,"=",H," * ",B,"+",M
```

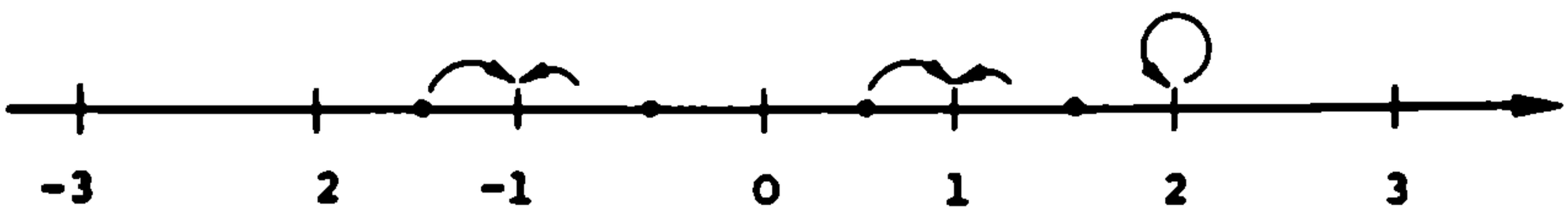
● Kerekítsük egész számra A változó értékét!

Az INT(X) *egészérték függvény* azt a legnagyobb számot adja, amelyik nem nagyobb, mint X értéke. Szemléletesen ez azt jelenti, hogy egy számhoz azt az egész számot rendeljük, amelyik tőle balra a legközelebb van a számegyenesen, ha a számot a számegyenesen ábrázoljuk (36. ábra).



36. ábra

A negatív számtartományban nem egyezik a FIX és az INT függvények értéke. Ábrázoljuk a számegyenesen a kerekítéssel történő hozzárendelést (37. ábra)!



37. ábra

Egy szám kerekítésekor azt az egész számot kell kapnunk, amelyik A értékéhez a legközelebb van. Egy egység annak az intervallumnak a hossza, amelyből választva a kerekítendő számot, ugyanazt az értéket kapjuk.

Az INT függvénnyel két egész szám közötti intervallum pontjaihoz rendeljük ugyanazt a számot. Helyezzük ide a kerekítendő számot. Adjunk a számhoz 0,5-et!

```
10  REM KEREKITES
20  INPUT "KEREKITENDO SZAM",A
30  A=INT(A+0.5)
40  PRINT "EGESZRE KEREKITVE",A
```

Amennyiben két tizedesre akarjuk kerekíteni a számot, a 30-as sor a következő lesz:

```
30 A = INT(A*100+0.5)/100
```

Az *eredményeket kerekítve* a PRINTUSING utasítással is kiírathatjuk. A # jelekkel, és a tizedesponttal meghatározhatjuk, hogy hány tizedes pontossággal írja ki a változó értékét. Az utolsó számjegyet kerekítve kapjuk meg.

A következő program két tizedes pontossággal és ahogyan a K$ változó értékével kijelöljük, úgy irja ki A értékét.

```
10  REM PRINT USING
20  INPUT "A SZAM",A
30  INPUT "FORMATUM",K$
40  PRINTUSING"####.##",A
50  PRINTUSING K$,A
```

A PRINTUSING további lehetőségeit a 4. mellékletben találjuk, a PRINTUSING utasítás leírásánál.

Kerekített értékek kiíratását a PRINTUSING utasítással egyszerűen megoldhatjuk, de ha kerekített értékkel szeretnénk számolni, akkor az INT függvénnyel kell a kerekítést elvégezni.

● Állítsunk elő program segítségével 5 db LOTTÓ számot!

Az, hogy milyen LOTTÓ számokat sorsolnak ki, a véletlenen múlik. Ilyen véletlen események szimulálását tudjuk programozni a RND függvény segítségével.

Véletlen számot állít elő az argumentumtól függően egy adott számtartományból.

RND (0) – a függvény értéke egy egyszeres pontosságú szám a (0; 1) intervallumból;

RND (N) – értéke egy egész szám az [1; N] intervallumból.

Az RND függvény első értékét a RANDOM utasítással állítjuk elő.

```
10  REM LOTTO SZAMOK
20  RANDOM
30  FOR I=1 TO 5
40  PRINT RND(90)
50  NEXT I
```

Másik lehetséges megoldás:

```
40  PRINT INT(90*RND(0)+1)
```

● Mozogjon a képernyőn egy karakter véletlenszerűen jobbra–balra!

Az RND függvénnyel +1-et és –1-et kell előállítanunk, hogy növeljük vagy csökkentsük a karakterpozíció értékét (ahová a karakter kerül). Használjuk fel a SGN függvényt, azaz az *előjel függvényt.*

Akkor, ha az argumentum pozitív, a függvény értéke 1, 0-nál 0, ha negatív, akkor –1. Vegyük a véletlen számokat a (–0,5; 0,5) intervallumból, akkor a SGN függvénnyel előállíthatjuk a kívánt értékeket. 0 esetén helyben marad a pont, de ennek nagyon kicsi a valószínűsége.

```
10  REM IMBOLYGO KARAKTER
20  CLS
30  RANDOM:P=32
40  Q=P+SGN(RND(0)-0.5)
50  PRINT@8*64,TAB(Q),"*"
60  PRINT@8*64,TAB(P)," "
70  P=Q:IF Q>=0 AND Q<64 THEN 40
```

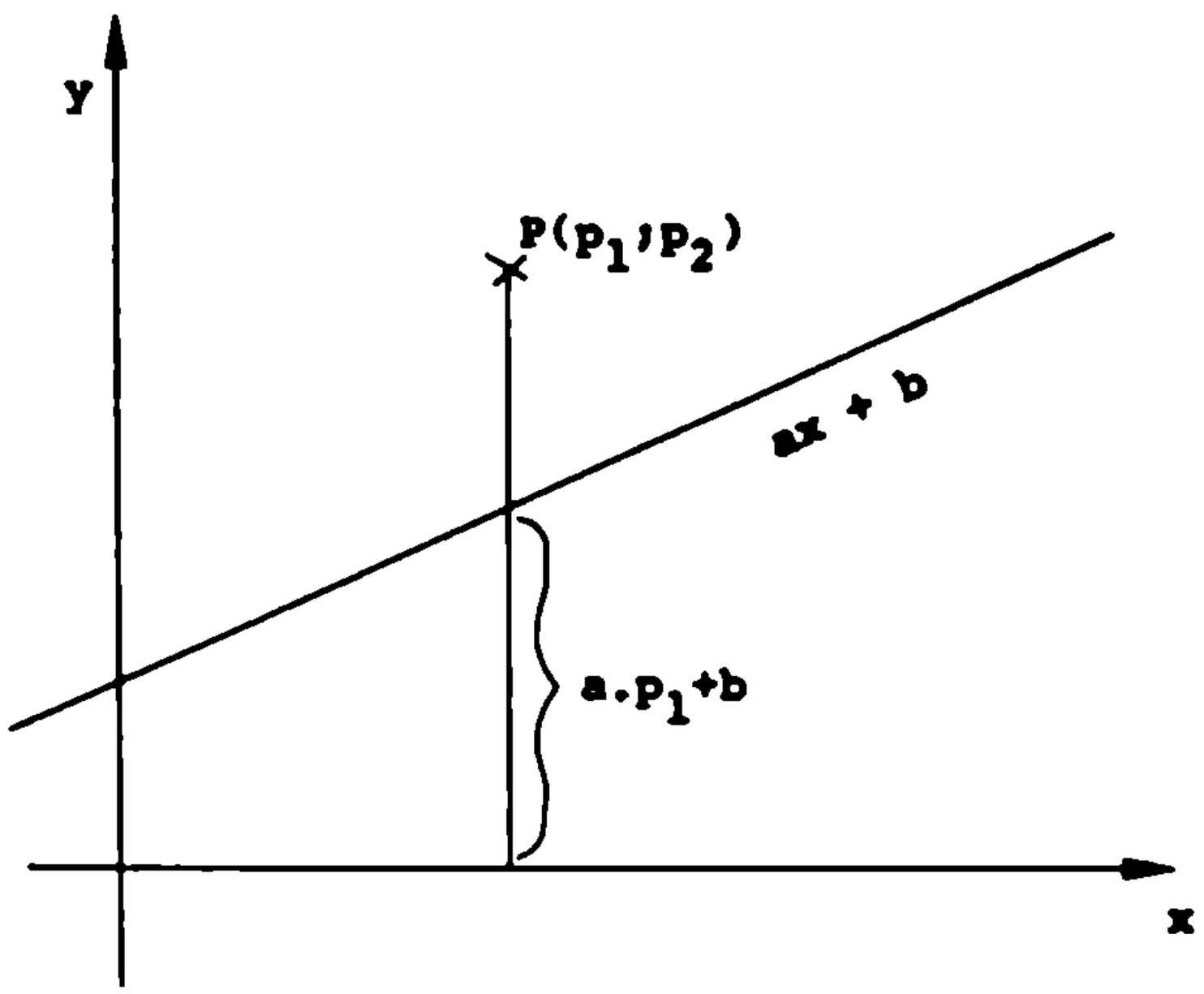
A PRINT @ utasítással a képernyő 8. sorának elejére állítottuk a kurzort. (A sorok sorszámozását a 0. sortól kezdtük.) A *TAB függvénnyel* abban a sorban jelölhetünk ki egy karakterpozíciót, ahol a kurzor áll. A TAB(Q) függvénnyel a kurzort a sor Q. karakterpozíciójába állítjuk, ahol megjelenítjük a karaktert. A 60-as sorban az előzőleg kiírt karaktert töröljük, a szóköz karakterrel.

A TAB függvény nem aritmetikai függvény. Az argumentummal *a képernyő egy sorának karakterpozícióját* jelöljük ki.

A TAB függvénnyel a kiíratás helyét jelölhetjük ki, amiből következik, hogy csak a PRINT utasításban használhatjuk.

• Adott az ax+b egyenes és a $P(p_1 ; p_2)$ pont. Írjunk programot, amivel meghatározhatjuk a pont és az egyenes kölcsönös helyzetét! A program írja ki, hogy ALATTA VAN vagy FELETTE VAN, vagy, hogy ILLESZKEDIK!

Az IF utasítás helyett alkalmazzuk az SGN függvényt, amivel ugyancsak megoldható a feladat. A 38. ábra a pont és az egyenes egy lehetséges elhelyezkedését mutatja.



38. ábra

A pont ordinátáját (p_2) kell összehasonlítanunk az egyenes p_1 abcisszájú pontjának ordinátájával ap_1+b. A $p_2-(ap_1+b)$ kifejezés értéke, a pont és az egyenes elhelyezkedésétől függően lehet negatív, 0 vagy pozitív. Az IF utasítással ezt kellene vizsgálni.

A $p_2-(ap_1+b)$ kifejezés SGN függvénye -1, 0 vagy 1 értékek közül valamelyiket veszi fel. 2-vel növelve ezt az értéket, 1-et, 2-t vagy 3-at kapunk.

Oldjuk meg a feladatot az ON–GOTO utasítással!

```
10  REM PONT-EGYENES-1.
20  CLS:INPUT "PONT KOORDINATAI",P1,P2
30  INPUT "A*X+B PARAMTEREI",A,B
40  Y=AXP1
50  ON SGN(P2-Y)+2 GOTO 70,80
60  PRINT "FELETTE VAN":STOP
70  PRINT "ALATTA VAN":STOP
80  PRINT "ILLESZKEDIK"
```

Ez a program azon kívül, hogy az SGN függvény alkalmazására látunk példát, arra is jó, hogy meggyőződjünk arról, hogy a számítógép csak egy megadott pontossággal számol. A programunk nem működik úgy, ahogyan azt várjuk. Az illeszkedést nem jelzi minden esetben, holott mi esetleg úgy adtuk meg az adatokat.

Írassuk ki a P2−Y kifejezés értékét, és akkor látni fogjuk, hogy ilyenkor az eredmény nem mindig 0.

Az *egyenlőséget nem lehet 0-ra vizsgálni.* Meg kell adnunk azt az értéket, aminél ha kisebb az eltérés, már azt mondjuk, hogy a pont illeszkedik az egyenesre. Legyen ez a határ 10^{-6}. A gyakorlatban ilyen eltérés már nem tekinthető számottevőnek.

```
10  REM PONT-EGYENES-2.
20  CLS:INPUT "PONT KOORDINATAI",P1,P2
30  INPUT "A*X+B PARAMTEREI",A,B
40  Y=A*P1
50  D=Y-P2
60  IF ABS(D)<1E-6 THEN PRINT
    "ILLESZKEDIK":END
70  IF D<0 THEN PRINT "FELETTE VAN"
    ELSE PRINT "ALATTA VAN"
```

● Adott egy egész szám. Irassuk ki a számjegyeket fordított sorrendben!

A szám felbontását számjegyekre az INT függvénnyel oldjuk meg. 10-zel osztva a számot és képezve az így kapott számnak a törtrészét, megkapjuk az utolsó számjegyből álló szám tizedrészét.

Azt, hogy a szám hány számjegyből áll, a szám tízes alapú logaritmusából határozható meg.

A *LOG függvény* az argumentum *természetes alapú logaritmusát* adja, aminek az alapszáma e = 2,71828. . . A függvény matematikai jelölése lnx. Egy szám tízes alapú logaritmusának kiszámításához használjuk fel a $\lg x = \frac{\ln x}{\ln 10}$ összefüggést.

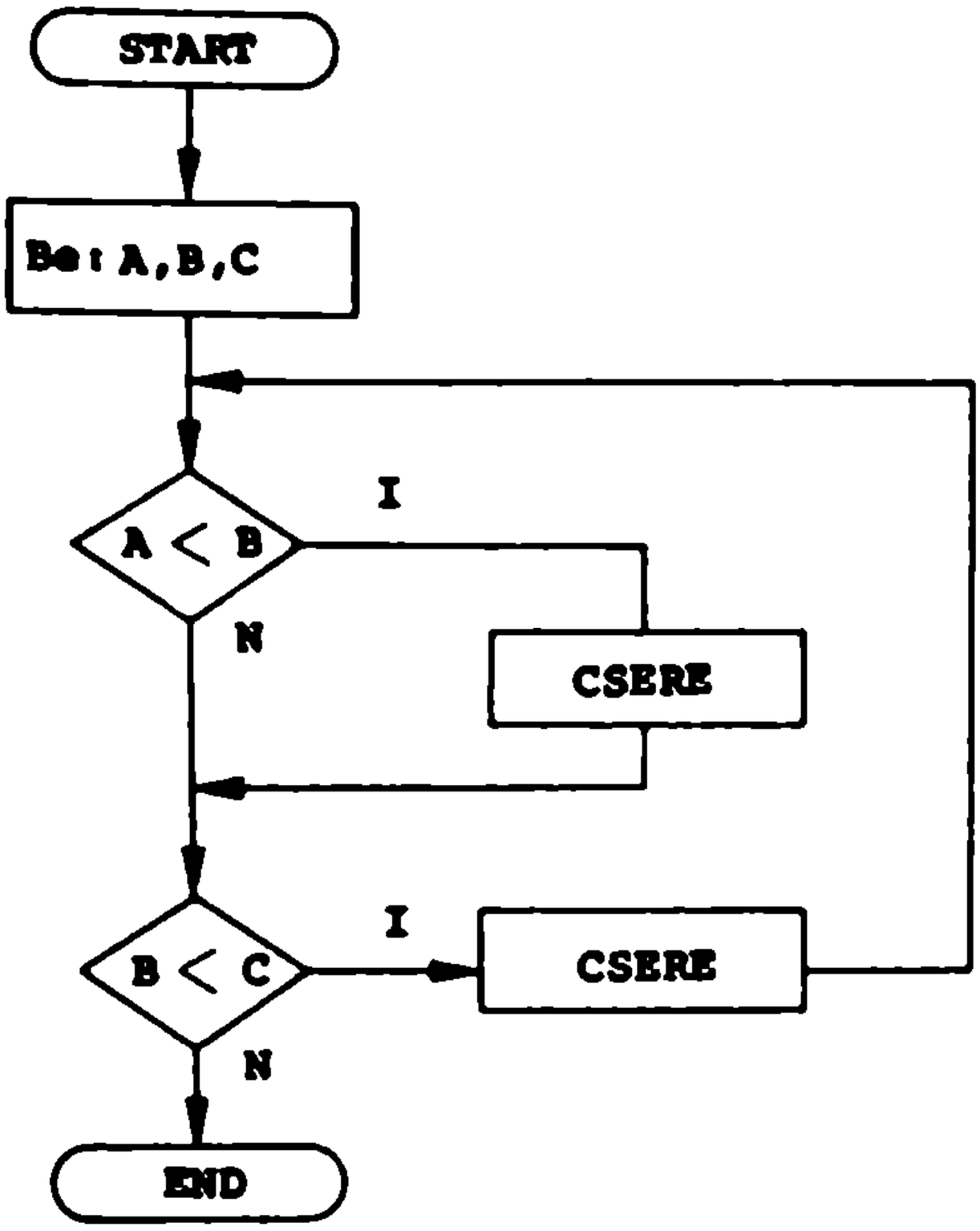
```
10  REM SZAM FORDITVA
20  INPUT "A SZAM",X
30  N=LOG(X)/LOG(10)+1
40  FOR I=1 TO N
50  Y=INT(X/10):T=10*(X/10-Y)
60  PRINT INT(T+0.5),
70  X=Y
80  NEXT I
```

● Adott három szakasz. Határozzuk meg a szakaszokból szerkesztett háromszög szögeit!

A három szakaszt nagyság szerint sorbarendezzük. Erre azért van szükség, hogy a legnagyobb oldallal szemközti szöget meg tudjuk határozni koszinusztétel segítségével. A következő szög, amit szinusztétellel számítunk ki, így biztos, hogy hegyesszög lesz.

Mielőtt a számításokat elkezdenénk, megvizsgáljuk, hogy a három szakaszból szerkeszthető-e háromszög. Miután sorbarendeztük a szakaszokat, elég azt megnézni, hogy a legnagyobb szakasz nagyobb-e, mint a két rövidebb szakasz összege.

A sorbarendezést a következő algoritmussal oldjuk meg (39. ábra):



39. ábra

A *trigonometrikus függvényekről* tudnunk kell, hogy az argumentumokat radiánban kell megadni és radiánban kapjuk a szöget is az ATN függvénnyel.

Ez az *arkusztangens függvény*, aminek az értéke az a szög lesz, amelyiknek a tangense az argumentummal egyenlő.

Egy szög szinuszát vagy koszinuszát ismerve, a szög kiszámításához a következő azonosságokat kell alkalmaznunk:

$$\mathrm{tg}x = \frac{\sin x}{\sqrt{1-\sin^2 x}} = \frac{\cos x}{\sqrt{1-\cos^2 x}}$$

```
 10  REM HAROMSZOG SZOGEI
 20  CLS:INPUT "A HAROM OLDAL",A,B,C
 30  IF A<B THEN T=A:A=B:B=T
 40  IF B<C THEN T=B:B=C:C=T:GOTO 30
 50  IF B+C<=A THEN 20
 60  X=(B↑2+C↑2-A↑2)/(2*B*C)
 70  X=SQR(1-X↑2)/X
 80  X=ATN(X)
 90  IF X<0 THEN X=4*ATN(1)+X
100  P=X*45/ATN(1)
110  Y=SIN(X)*B/A:Y=ATN(Y/SQR(1-Y↑2))
120  Q=Y*45/ATN(1)
130  Z=4*ATN(1)-X-Y:R=180-P-Q
140  R$="FOK":F$="RADIAN"
```

```
150  PRINT "ALFA",X,F$,P,R$
160  PRINT "BETA=",Y,F$,Q,R$
170  PRINT "GAMMA=",Z,F$,R,R$
```

- Igaz az, hogy

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Készítsünk programot, amivel meghatározhatjuk egy szög szinuszát adott pontossággal!

Az összeg első tagja x. Az i. tag ismeretében az i+1. tag:

$$a_{i+1} = -a_i \frac{x^2}{(2i-2)(2i-1)}$$

Az így képzett tagokat összegezzük mindaddig, ameddig valamelyik tag abszolútértéke kisebb nem lesz, mint a megadott pontosság, azaz nem befolyásolja az összeg értékét a pontosságnál nagyobb mértékben.

```
10  REM SZOG SZINUSZA
20  DEFDBL A-Z:CLS:INPUT "SZOG RADIANBAN",X
30  INPUT "PONTOSSAG",E
40  Y=X:A=X:I=1
50  I=I+1
60  A=-A*X*X/(2*I-2)/(2*I-1)
70  Y=Y+A
80  PRINT@ 580,"SIN",X,"=",Y
90  IF ABS(A)>E THEN 50
```

Kérdések és feladatok

1. Ábrázoljuk és elemezzük az aritmetikai függvényeknek megfelelő elemi függvényeket! Határozzuk meg értelmezési tartományukat és az értékkészletüket!
2. Határozzuk meg az egész számoknak azt a részhalmazát, amit
   a) az egész,
   b) az egyszeres pontosságú,
   c) a kétszeres pontosságú változók értékei alkotnak!
3. Határozzuk meg a valós számoknak azt a részhalmazát, amit
   a) az egyszeres pontosságú,
   b) a kétszeres pontosságú változók értékei alkotnak!
4. Milyen közelítő eljárást ismerünk még a számok négyzetgyökének meghatározására?
5. Hogyan oldható meg a maradékos osztás a FIX függvény használata nélkül?
6. Hasonlítsuk össze a FIX és az INT függvényt!
7. Végezzük el a maradékos osztást az INT függvény alkalmazásával!
8. Kerekítsük a számokat a FIX függvény segítségével!
9. Írassuk ki a négyzetgyök közelítő értékét a megadott pontossággal!
10. Állapítsuk meg a FIX, az INT és a CINT függvények értelmezési tartományát!
11. Készítsünk „kockadobó" programot!

12. Írjunk programot, amivel az összeadást lehet gyakorolni az egész számok körében!

13. Írjuk át az IMBOLYGO KARAKTER programot úgy, hogy

a) ne használjuk a TAB függvényt;

b) minden irányban mozoghasson a karakter;

c) kétszer olyan valószínűséggel mozogjon balra, mint jobbra!

14. Írjunk programot, amivel azt játszhatjuk, hogy az „imbolygó karaktert" egy másik karakterrel „elfogjuk"!

15. Határozzuk meg egy szám n. gyökét a LOG és EXP függvények segítségével!

16. Készítsük el a SZAM FORDITVA program folyamatábráját!

17. Ellenőrizzük a HAROMSZOG SZOGEI programot! Vizsgáljuk meg, hogy speciális vagy elfajuló esetekben helyesen működik-e a program! Nézzük meg az összes lehetséges esetet!

18. Határozzuk meg a csúcspontjainak koordinátáival adott háromszög kerületét, területét, szögeit és nevezetes pontjainak koordinátáit program segítségével!

19. Készítsünk programot a kör és a pont kölcsönös helyzetének vizsgálatára!

20. Készítsünk TOTO tippeket előállító programot!

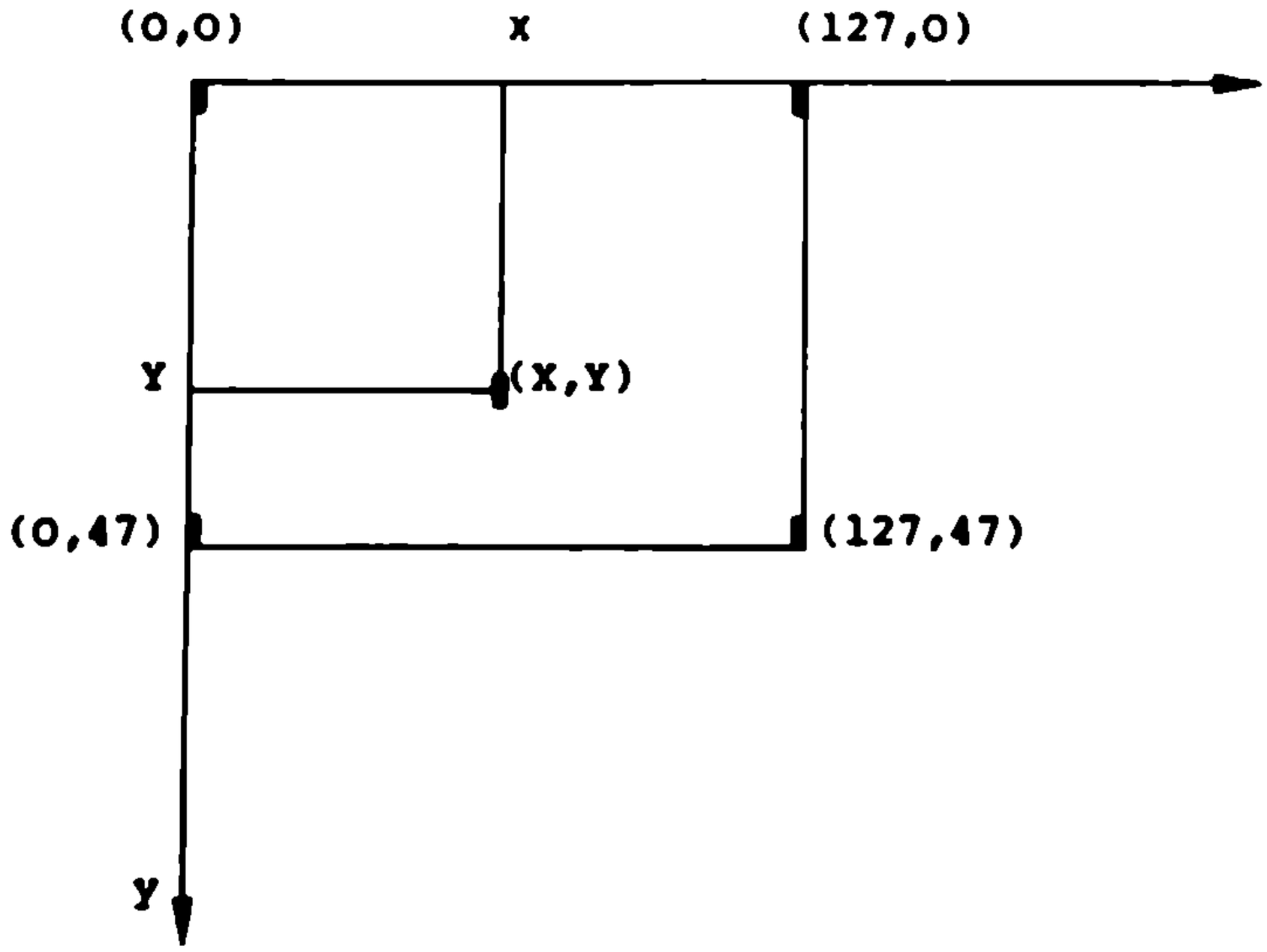
21. Határozzuk meg a következő közelítő értékeket 1E–6 pontossággal:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

GRAFIKA

A képernyőt tekintsük egy olyan koordináta-rendszernek, amelyen az első negyed egy részének rácspontjait BASIC utasítással megjeleníthetjük, törölhetjük. A képernyő koordináta-rendszerének Y tengelye lefelé mutat (40. ábra).



40. ábra

A SET (X,Y) utasítással *megjelenítjük*, a RESET(X, Y) utasítással *töröljük a képernyő* (X,Y) koordinátájú *pontjdt.*

A 40. ábrán a képernyő sarokpontjainak koordinátáit adtuk meg, amiből arra következtethetünk, hogy a

$0 \leq X \leq 127$

$0 \leq Y \leq 47$

relációknak teljesülniük kell.

A pont koordinátáit tetszőleges aritmetikai kifejezéssel megadhatjuk, ezért szintaktikailag helyesek a következő utasítások: SET(A+1,15) RESET(I,2*J) SET(X,20*SIN(X)+24). Ezeknek a kifejezéseknek az egészértéke lesz a pont koordinátája.

- Mozogjon egy világító pont véletlenszerűen a képernyőn!

A feladatban azt kell megoldanunk, hogy az (X,Y) koordinátájú pont megjelenítése után valamelyik szomszédos pontot világítsuk ki, és az előző pontot töröljük. A pont és a ponttal szomszédos pontok koordinátái (41. ábra):

X-1,Y-1	X,Y-1	X+1,Y-1
X-1,Y	X,Y	X+1,Y
X-1,Y+1	X,Y+1	X+1,Y+1

41. ábra

Vegyük észre, hogy a szélső pontok koordinátái azonosak vagy 1-gyel nagyobbak vagy kisebbek a középső pont koordinátáinál. A 0, 1 és −1 számokat kell előállítanunk azonos valószínűséggel, az RND függvény segítségével. Így kapjuk a pont újabb koordinátáit.

A pont előző koordinátáinak értékét tároljuk, hogy azt törölni tudjuk.

```
10  REM BOLYONGO PONT-1
20  RANDOM:CLS
30  X1=RND(128)-1:Y1=RND(48)-1
40  K=RND(3)-2:L=RND(3)-2
50  X=X1+K:Y=Y1+L
60  IF X<0 OR X>127 OR Y<0 OR Y>47 THEN END
70  SET(X,Y):RESET(X1,Y1)
80  X1=X:Y1=Y:GOTO 40
```

A 60-as sorban azt vizsgáljuk, hogy a pont a képernyőre esik-e. Negatív vagy az adott határoknál nagyobb koordináták esetén a rendszer hibát jelezne.

Kihasználva azt, hogy a pont bármelyik koordinátája legfeljebb 1-gyel változik, a vizsgálatot elvégezhetjük a következő utasítással is:

```
60  IF(X+1)*(Y+1)*(X-128)*(Y-48)=0 THEN END
```

• A „bolyongó pont" mozgását tegyük szemléletesebbé azáltal, hogy a pontot csak akkor jelenítjük meg, ha a képernyőnek ez a pontja sötét volt előzőleg! Amennyiben világos volt, akkor töröljük a pontot!

A képernyő adott pontjának állapotától függ, hogy mi történik. A POINT(X,Y) *függvénnyel vizsgálhatjuk* meg azt a pontot, ahová a „bolyongó pont" kerül. Logikai függvényként értelmezve a POINT függvény értéke IGAZ, ha világít a pont, HAMIS, ha a képernyő X,Y koordinátájú pontja sötét. Ennek megfelelően a függvény aritmetikai értéke −1, ill. 0.

```
10  REM BOLYONGO PONT-2
20  RANDOM:CLS
30  X=RND(128)-1:Y=RND(48)-1
40  X=X+RND(3)-2:Y=Y+RND(3)-2
50  IF X<0 OR X>127 OR Y<0 OR Y>47 THEN END
60  IF POINT(X,Y)THEN RESET(X,Y)
70  IF POINT(X,Y)=0 THEN SET(X,Y)
80  GOTO 40
```

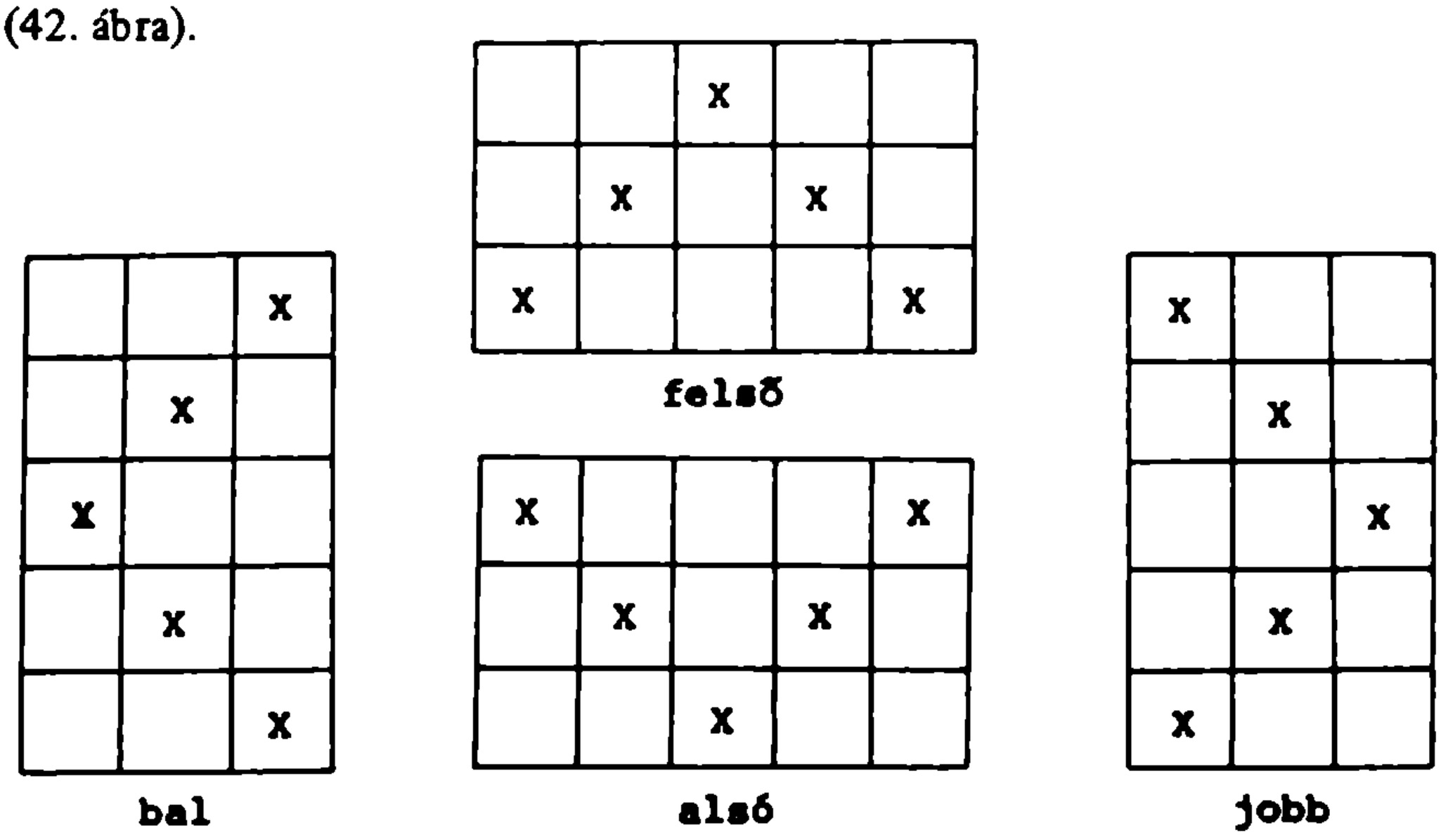
A programban kétféleképpen is megfogalmazzuk a pont állapotára vonatkozó ítéletünket az IF utasításokban (60-as és 70-es sor).

A két IF utasítás helyett alkalmazhatjuk az IF POINT(X,Y) THEN RESET(X,Y) ELSE SET(X,Y) utasítást is.

• Pattogjon egy golyó négy fal között! A falakat a képernyőre rajzolt egyenesekkel jelezzük. A golyót a képernyő egy világító pontjának mozgásával szimuláljuk. A falak kirajzolásakor csak az egyik koordinátát kell változtatni, a másik állandó.

A mozgó pont pályáját úgy rajzolhatjuk meg, ha az ütközés 45°-os, hogy a vízszintes koordináták egységnyi változtatásával a függőleges koordinátát is egységnyivel változtatjuk.

A változást V és F változók értékeivel adjuk meg. Kezdőértéküket az RND függvénnyel állítjuk elő (80-as sor). Értékük 1 vagy −1 lehet. Ezek az értékek két ütközés között nem változnak. Vizsgáljuk meg a programban, hogy melyik falnak ütközött a pont, ebből meghatározható, hogy a koordináták értékei hogyan változnak az ütközés után (42. ábra).



42. ábra

Ütközés az oldalsó falba:
– X értékének a változása ellentétes lesz: V = –V.
Ütközés az alsó vagy felső falba:
– Y értéke változik ellentétesen: F = –F.

```
 10  REM PATTOGO GOLYO
 20  CLS:RANDOM
 30  REM BAL-JOBB FAL
 40  FOR I=0 TO 47:SET(0,I):SET(127,I):NEXT
 50  REM ALSO-FELSO FAL
 60  FOR I=0 TO 127:SET(I,0):SET(I,47):NEXT
 70  REM A PONT
 80  X1=RND(126):Y1=RND(46)
 90  REM MOZGAS IRANY
100  V=SGN(RND(0)-0.5):F=SGN(RND(0)-0.5)
110  X=X1+V:Y=Y1+F
120  IF X=1 OR X=126 THEN V=-V
130  IF Y=1 OR Y=46 THEN F=-F
140  SET(X,Y):RESET(X1,Y1)
150  X1=X:Y1=Y:GOTO 110
```

• Írjunk programot, amivel nagyobb formátumban jeleníthetjük meg a betűket! A betűket 7 soros és 5 oszlopos pontmátrixszal adjuk meg (43. ábra).

		X		
	X		X	
X				X
X				X
X	X	X	X	X
X				X
X				X

43. ábra

A programban a DATA utasítással valamilyen módon meg kell adnunk, hogy az egyes betűknél a pontmátrix melyik pontját kell megjeleníteni. A pontmátrix pontjainak megadása többféleképpen történhet. Az a mód, ahogyan az adatok adottak, meghatározza a megjelenítő eljárást is.

A programok írása *adatszerkezetek kialakítását*, és *algoritmusok megalkotását* jelenti. A két tevékenység elválaszthatatlan egymástól.

a) Jelöljük a pontmátrix pontjainak állapotát az 1 és a 0 számjegyekkel, attól függően, hogy a pont világít vagy nem!

Minden programban csak az A betűt kódoljuk. A program kiegészítése a többi betűvel már nem okozhat gondot.

```
 10  REM BETUK-1
 20  CLS:INPUT "BETU",B$
 30  CLS:U=50:V=20
 40  REM KIKERESES
 50  READ X$:IF B$<>X$ THEN 50
 60  REM DEKODOLAS-MEGJELENITES
 70  FOR I=1 TO 7:FOR J=1 TO 5
 80  READ X
 90  IF X=1 THEN SET(U+J,V+I)
100  NEXT J:NEXT I
200  DATA A,0,0,1,0,0
210  DATA 0,1,0,1,0
220  DATA 1,0,0,0,1
230  DATA 1,0,0,0,1
240  DATA 1,1,1,1,1
250  DATA 1,0,0,0,1
260  DATA 1,0,0,0,1
```

b) A pontmátrix sorait bináris számokkal adjuk meg. A számjegyek jelzik a pontok állapotát. A szám számjegyekre történő felbontását programmal oldjuk meg.

A megjelenítendő betű kérése és kikeresése az előző program szerint történik, ezért csak a kódolást, a megjelenítést és az adatábrázolást írjuk le.

```
 60  REM DEKODOLAS-MEGJELENITES
 70  FOR I=1 TO 7
 80  READ X
 90  FOR J=1 TO INT(LOG(X)/LOG(10))+1
100  P=X-10*INT (X/10)
110  IF P=1 THEN SET(U+5-J,V+I)
120  X=(X-P)/10
130  NEXT J:NEXT I
140  DATA A,100,1010,10001,10001
150  DATA 11111,10001,10001
```

Az előző programokban mindig adott számú adatot kellett beolvasni egy betű megjelenítéséhez. Ez lehetővé tette, hogy a betűket karakterrel adhattuk meg.

A következő programokban a *betűk kódját* kérjük. Kódjuk alapján keressük ki az adatokat. A karakterek kódját a 8. melléklet tartalmazza.

c) Adjuk meg a DATA adatmezőben a pontmátrix azon pontjainak sor- és oszlopszámát, amelyek világítanak!

A kódszámokat úgy különböztetjük meg a világító pontok helyzetét meghatározó adatoktól, hogy azokat negatív számokkal adtuk meg. Ez jelzi a következő betűhöz tartozó adatok kezdetét.

```
 10  REM BETUK-3
 20  CLS:INPUT "KARAKTER KODJA",K
 30  REM HELYKIJELOLES
 40  CLS:U=50:V=20
 50  REM KIKERESES
 60  READ X:IF -X<>K THEN 60
 70  REM DEKODOLAS
 80  READ I,J:IF I<0 THEN END
 90  SET(U+J,V+I):GOTO 80
100  DATA -65,1,3,2,2,2,4,3,1,3,5,4,1
110  DATA 4,5,5,1,5,2,5,3,5,4,5,5,6,1
120  DATA 6,5,7,1,7,5
130  DATA -66
```

d) A világító pontok két koordinátáját megadhatjuk egy kétszámjegyű számmal is. A számjegyek a sort és az oszlopot jelölik. A szétválasztást a program végzi.

```
 80  READ P:IF P<0 THEN END
 90  I=INT(P/10):J=P-10*I
100  SET(U+J,V+I):GOTO 80
200  DATA -65,13,22,24,31,35,41
210  DATA 45,51,52,53,54,55,61,
220  DATA 65,71,75
230  DATA -66
```

A karakterlánc-függvények ismeretében (6. melléklet) megírhatjuk az előző két programot úgy is, hogy a betű kódja helyett a betűket írjuk.

e) Adjuk meg soronként egy-egy decimális számmal a világító pontokat. A szám számjegyei jelezzék, hogy hányadik pont világít a sorban.

```
 10  REM BETUK-5
 20  CLS:INPUT "KARAKTER KODJA",B$
 30  REM HELY KIJELOLES
 40  CLS:U=50:V=20
 50  REM KIKERESES
 60  READ X$:IF X$<>B$ THEN 60
 70  REM DEKODOLAS
 80  FOR I=1 TO 7
 90  READ S
100  T=INT(S/10):J=S-10*T
110  IF J=0 THEN 130
120  SET(U+J,V+I):S=(S-J)/10:GOTO 100
130  NEXT
140  END
150  DATA A,3,24,15,15,12345,15,15
```

- Rajzoljuk ki programmal a képernyőre a $\frac{\sin 2x}{x}$ függvényt!

A feltételek legyenek a következők:

– legyen adott annak az intervallumnak a kezdőpontja, amelyben ábrázoljuk a függvényt (X értéke);

– adjuk meg azt, hogy milyen lépésközönként ábrázoljuk a függvényt (D értéke);

– a tengelyeken két pont távolsága D értékével legyen egyenlő.

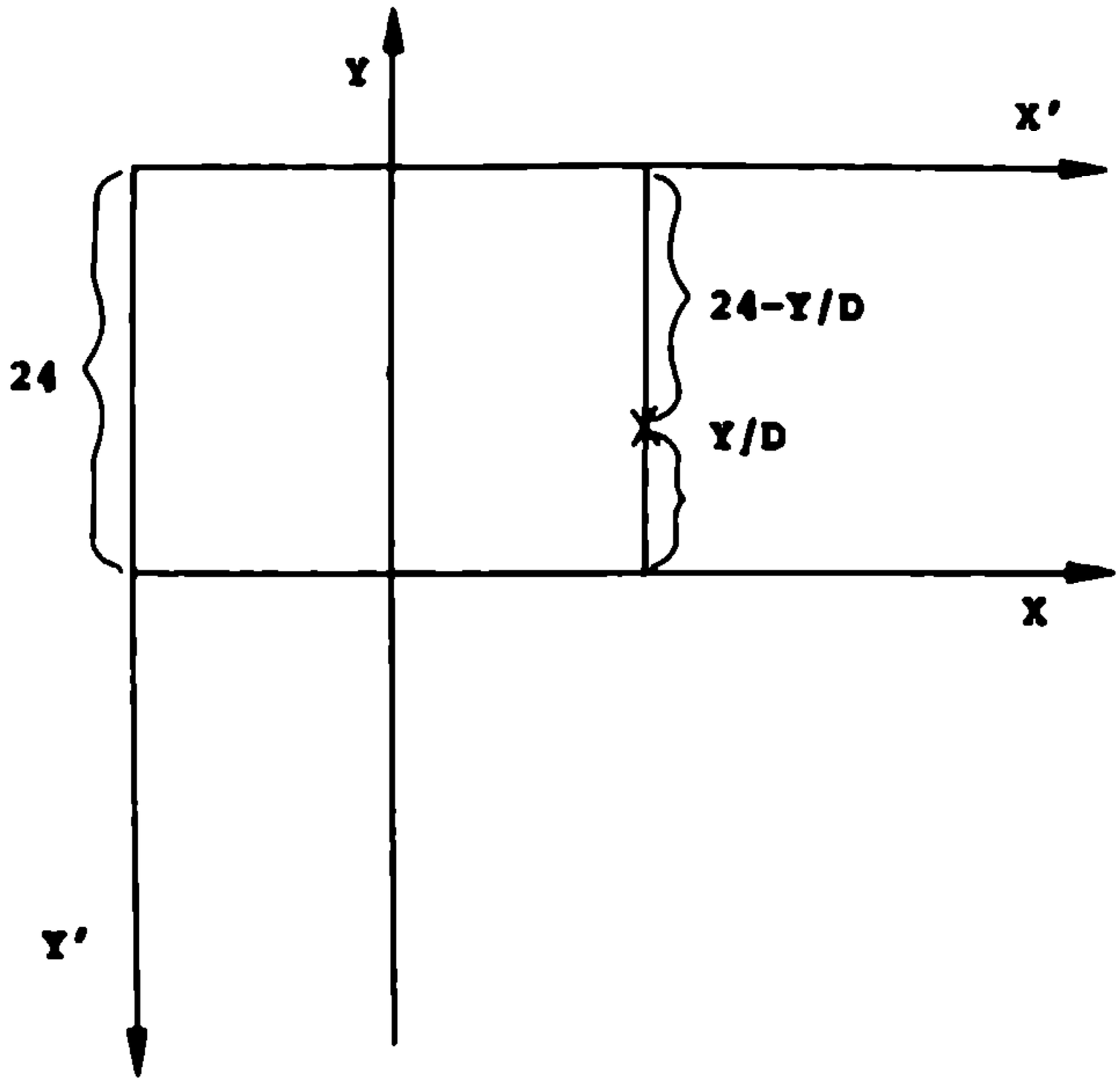
Két szomszédos pont távolsága a tengelyen azonos.

```
 10  REM FUGGVENY-1
 20  CLS:INPUT "KEZDOERTEK",X
 30  INPUT "LEPESKOZ",D:CLS
 40  REM KOORDINATA TENGELYEK
 50  FOR I=0 TO 127:SET(I,24):NEXT
 60  IF X>0 OR -X/D>127 THEN 90
 70  FOR I=Ø TO 47:SET(-X/D,I):NEXT
 80  REM FUGGVENY RAJZOLAS
 90  FOR I=Ø TO 127
100  IF X=Ø THEN 140
110  Y=SIN(2*X)/X
120  Y=-Y/D+24:Y=INT(Y+0.5)
130  IF Y>=0 AND Y<48 THEN SET(I,Y)
140  X=X+D
150  NEXT
160  IF INKEY$=""THEN 160 ELSE END
```

Először kirajzoljuk az X tengelyt (50-es sor) a képernyő közepére. A 60-as sorban azt vizsgáljuk, hogy ki kell-e rajzolni az Y tengelyt.

A függvény kirajzolása úgy történik, hogy X kezdőértékétől kiindulva, D lépésközönként kiszámítjuk a függvény értékét 128 helyen. Az így kapott (X,Y) koordinátájú pontokat jelenítjük meg azután a képernyő koordináta-rendszerében úgy, ahogyan azt a feltételek előírják. A függvény értékét egy adott X értéknél a 100-as és a 110-es sorokban határozzuk meg. Az értelmezési tartomány vizsgálatát is ide kell számítanunk (100-as sor).

Az Y = Y/D + 24 utasítással transzformáljuk a függvényt a képernyő koordináta-rendszerébe, amelynek a tengelyeit x', y'-vel jelöltük a 44. ábrán. Az Y/D transzformációval elértük azt, hogy két pont távolsága függőlegesen is az az érték legyen, mint vízszintesen.



44. ábra

Az Y értékének kerekítésével kisebb lesz az ábrázolás hibája. A kerekítést természe-
.tesen megoldhatjuk a SET(I,Y+0.5) utasítással is.

● Ábrázoljuk a függvényt olyan koordináta-rendszerben, amelyiknek X tengelye függõlegesen helyezkedik el a képernyõn!

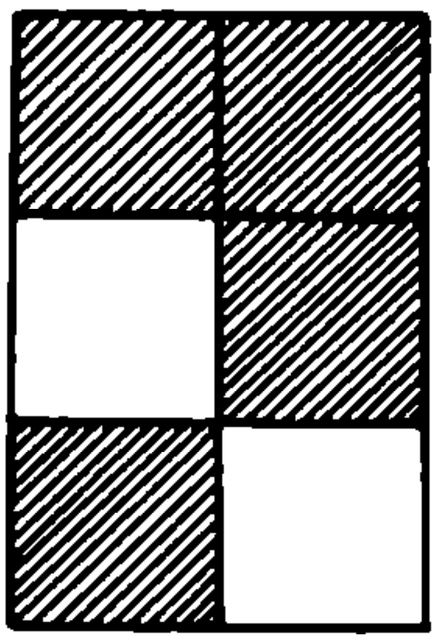
A PRINT utasitással rajzoljuk ki a függvényt. A függvényértéknek megfelelõ karakterpozícióba egy *-ot írunk. Írógépen vagy nyomtatón történõ függvényrajzolás legtöbbször ezzel a módszerrel történik.

Futtatva a programot, azt tapasztaljuk, hogy a függvényrajzolás folyamatos lesz. Az utolsó sorba írjuk a *-ot (nincs utána pontosvessző), ezután soremelés történik. A képernyő tartalma egy sorral feljebb kerül, és az utolsó sor üres marad (ide írhatjuk a következõ *-ot).

```
10  K=-10:V=10:D=0.2:CLS:A=20
20  FOR X=K TO V STEP D
30  PRINT@ 15*64+32,".",
40  PRINT@ 15*64+32+SIN(2*X)*A,"X"
50  NEXT
```

Ismerkedjünk meg a grafikus karakterekkel!

Egy karakterpozíció hat grafikus pontnak felel meg, amik egy grafikus karaktert alkotnak (45. ábra). A képernyõ tartalma tehát három képponttal kerül feljebb, ha az utolsó sorba írunk egy karaktert, ami után soremelést hajtunk végre.



45. ábra

```
10  REM GRAFIKUS KARAKTER
20  INPUT "SOR,OSZLOP";K,L
30  V=3*K:U=2*L :CLS
40  FOR I=0 TO 2:FOR J=0 TO 1
50  READ P
60  IF P=1 THEN SET(U+J,V+I)
70  NEXT:NEXT
90  DATA 1,1,0,1,1,0
```

A következő programban a képernyő utolsó sorában jelenítjük meg a függvénygörbe 3 pontját, majd a PRINT @1023, utasítással sort emelünk.

```
 10  REM FUGGVENY-2
 20  K=0.1:V=100:L=0.03
 30  CLS:J=0
 40  FOR X=K TO V STEP L
 50  Y=SIN(2*X)/X
 60  Y=Y/L+64
 70  Y=INT(Y+0.5)
 80  IF Y<0 OR Y>127 THEN 100
 90  SET(Y,45+J)
100  SET(64,45+J)
110  J=J+1
120  IF J=3 THEN PRINT@1023,:J=0
130  NEXT
```

Kérdések és feladatok

1. Ismertessük a grafikus utasításokat!
2. Jelenítsük meg egy adott pont szomszédos pontjait!
3. Oldjuk meg az AND művelettel annak vizsgálatát, hogy az (X,Y) koordinátájú pont a képernyőn van-e!
4. Jelenítsünk meg a képernyőn egy ábrát, majd annak negatív képét!
5. "Bolyongjon" világos mezőben egy sötét pont!
6. A "pattogó golyó" ütközési szöge ne 45°-os legyen!
7. Irjunk programot a cirill betűk megjelenítésére!
8. Ábrázoljunk más függvényeket is a tankönyv programjaival!
9. Mi a szerepe az 160 IF INKEY$=""THEN 160 ELSE END utasításnak a FUGGVENY–1 programban?
10. Mennyi lesz két pont távolsága az y tengely mentén a FUGGVENY–1 programban, ha az Y = –Y/D+24 utasítás helyett az Y = –Y + 24 utasítást írjuk?
11. Készítsünk függvénygörbe rajzoló programokat!
    a) Ábrázoljuk a függvényt egy adott intervallumban azonos beosztású tengelyekkel!

b) Adott értékhatárok között jelenítsük meg a függvényértékeket ($A \leq Y \leq B$)!
c) Legyen A a legkisebb, B a legnagyobb függvényértékkel egyenlő!

12. Miért kell kerekítést végezni a megjelenítés előtt?
13. Módosítsuk a következő programot úgy, hogy kör jelenjen meg a képernyőn!

```
10  CLS:INPUT "HANY PONT",B
20  FOR X=0 TO 8*ATN(1)STEP 0.1
30  SET(64+20*COS(X)+0.5,24+B*SIN(X)+0.5)
40  NEXT
50  IF INKEY$="" THEN 50
```

INDEXES VÁLTOZÓK

● Tízes alapú számrendszerben adott számot alakítsunk át nyolcas alapú számrendszerbe!

Végezzünk el egy ilyen átalakítást! Legyen az átalakítandó szám 1984.

	1984 : 8 = 248	248 : 8 = 31	31 : 8 = 3	3 : 8 = 0
maradékok:	0	0	7	3

A maradékos osztás maradékai adják a nyolcas számrendszerben a szám számjegyeit. Ezeket fordított sorrendben kellene kiíratnunk, hogy megkapjuk a szám nyolcas számrendszerbeli alakját: $1984_{10} = 3700_8$.

A számjegyeket tehát tárolnunk kell, hogy a kiírást az utoljára kapott számjeggyel kezdhessük.

A maradékokat a B *indexes változók* értékeként őrizzük meg a B(I) = X –8*H utasítással. Az I változó értékét a programban változtatjuk, és így 1984 átszámítása a következő utasításssorozatnak felel meg:

```
B(0)=0:B(1)=0:B(2)=7:B(3)=3
```

A *változó neve* után zárójelben adjuk meg, hogy hányadik számjegyről van szó. Ez lesz a *változó indexe.*

```
 10  REM KONVERTALAS NYOLCAS SZAMRENDSZERBE
 20  INPUT "TIZES SZAMRENDSZERBEN A SZAM",X
 30  I=0
 40  H=INT(X/8)
 50  B(I)=X-8*H
 60  X=H:I=I+1
 70  IF X<>0 THEN 40
 80  PRINT "NYOLCAS SZAMRENDSZERBEN",
 90  FOR J=I-1 TO 0 STEP -1
100  PRINT B(J),
110  NEXT J
```

A B indexes változók értékét a program végén íratjuk ki. A változókra indexükkel hivatkozunk, tehát számokkal. A kiíró ciklus visszafelé számlál, és ezzel megadjuk az indexes változó indexeinek értékét.

Az indexek természetes számok, ezért az indexes változók értékadását tekinthetjük olyan megfeleltetésnek, amikor a természetes számokhoz rendeljük egy számsorozat elemeit. Matematikai jelöléssel a számsorozatunk: $b_0 = \emptyset, b_1 = \emptyset, b_2 = 7, b_3 = 3$.

A számsorozat elemei rendezettek, mivel a természetes számok rendezett halmazt alkotnak. Ez indokolja, hogy a B indexes változót nevezhetjük B *vektornak* is.

● Adott egy N elemű számsorozat. Válasszuk ki a számsorozat legnagyobb és legkisebb elemét!

Az N darab számot beolvassuk az A vektorba. A vektor elemeire indexükkel hivatkozhatunk, amit egy változó értékeként adunk meg.

Feladatunkban egy vektor elemeit össze kell hasonlítanunk a korábban kiválasztott legnagyobb, ill. legkisebb elemmel. Ezt megoldhatjuk egy ciklussal, amivel az indexet változtatjuk.

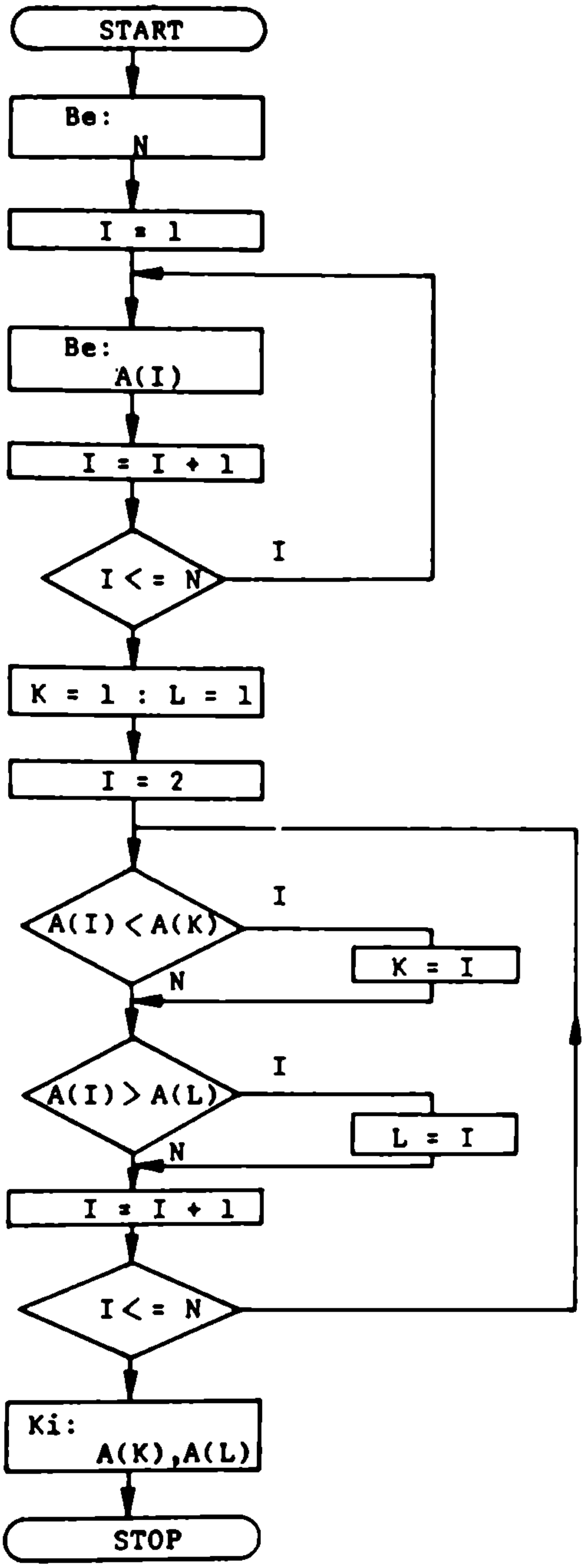
A K és az L változó a legnagyobb, ill. a legkisebb elem indexét jelenti. Az összehasonlítás a második elemtől kezdődik, mivel a legnagyobbnak, ill. a legkisebbnek az első elemet adjuk meg. K és L változó értéke a ciklusbalépés előtt 1 lesz (46. ábra).

Az indexes változók értékei a memóriában egymás után helyezkednek el. A *tároláshoz szükséges memóriaterületet* a DIM utasítással határozzuk meg. Így adjuk meg, hogy hány elemből áll a vektor. Az előző programban azért nem használtuk a DIM utasítást, mert 10 elem részére a rendszer mindenképpen biztosít helyet.

```
10  REM LEGKISEBB-LEGNAGYOBB
20  INPUT "ELEMEK SZAMA",N
30  DIM A(N)
40  FOR I=1 TO N:INPUT A(I):NEXT
50  K=1:L=1
60  FOR I=2 TO N
70  IF A(I)<A(K)THEN K=I
80  IF A(I)>A(L)THEN L=I
90  NEXT I
100 PRINT "LEGKISEBB=",A(K)
110 PRINT "LEGNAGYOBB=",A(L)
```

● Írjunk programot, amivel abc sorrendbe rendezhetünk angol szavakat!

A szavak számát előre nem ismerjük. A szavak bevitelét a & karakterrel zárjuk. Feltesszük, hogy 100-nál nincs több szó. A DIM utasítással az indexhatárt 100-nak veszszük. Ezzel megadjuk az elemek maximális számát.



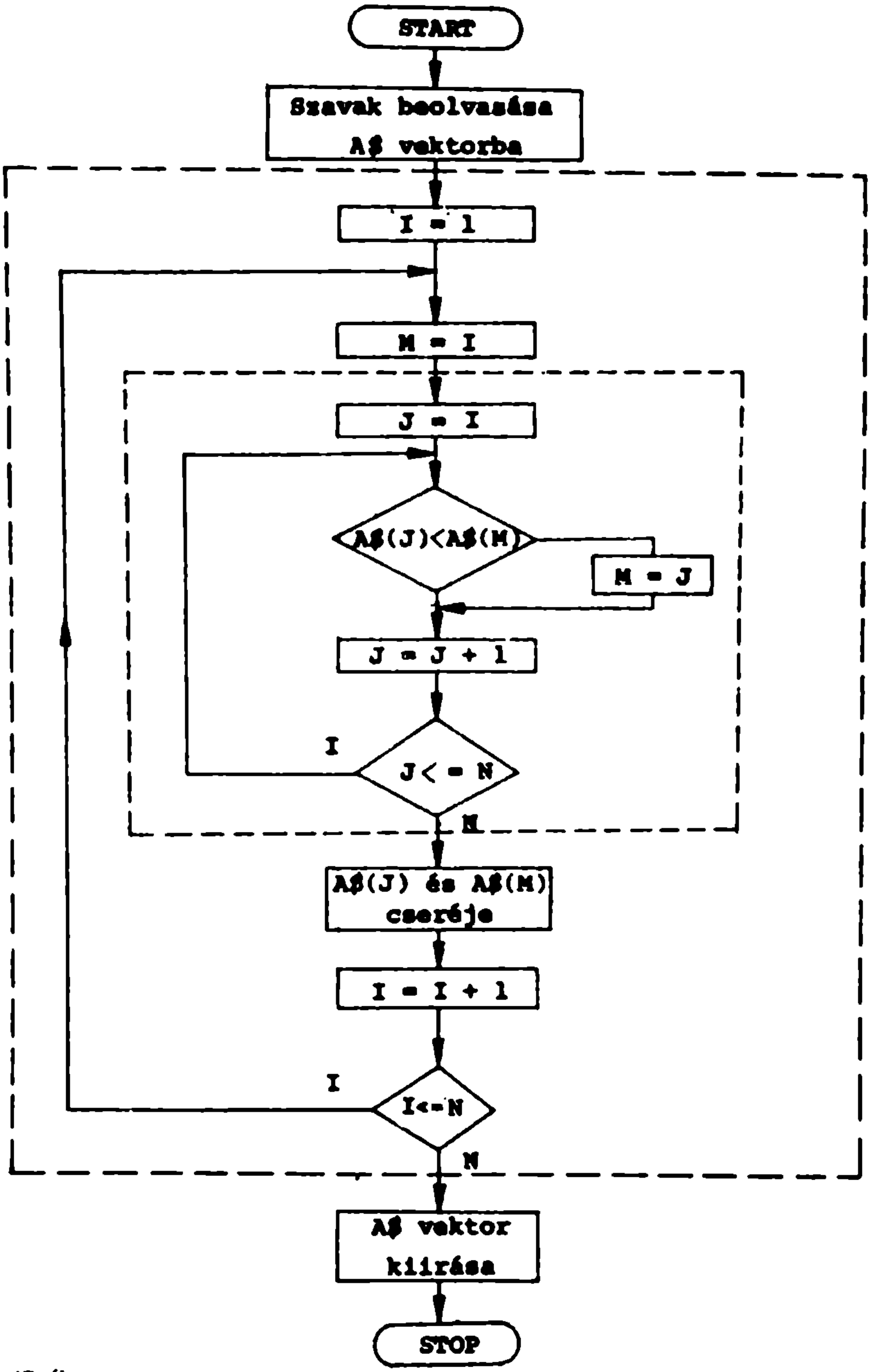
46. ábra

Karakterláncok esetében a karakterláncot alkotó karakterek számától függ, hogy mennyi hely szükséges a memóriából. Nagy pazarlás lenne, ha elemenként 255 karakter számára biztosítanánk helyet (ennyi karakterből állhat maximálisan egy karakterlánc).

Egy karakter elhelyezéséhez egy memóriarekeszre van szükség. Átlagban számoljunk a programunkban 8 betűs szavakkal. Ekkor 800 rekeszre van szükségünk a szavak

elhelyezésére. Ezt a memóriaterületet a CLEAR 800 utasítással foglaljuk le. Ez az utasítás törli a változók értékeit is, ezért általában a program elejére írjuk.

Két karakterlánc összehasonlításkor az a kisebb, amelyik az abc sorrendben meg-



47. ábra

előzi a másikat. A sorbarendezéshez a legkisebb elem kiválasztásának algoritmusát használjuk fel. Kiválasztjuk az elemek közül a legkisebbet, és azt a helyre tesszük.

A külső ciklussal nem csak azt adjuk meg, hogy a legkisebb keresése hányadik indextől kezdődjön, hanem azt is, hogy hová helyezzük el a legkisebb elemet. A legkisebb elem kikeresése mindig 1-gyel nagyobb indexszel kezdődik. A korábban kiválasztott elemeket így nem vizsgáljuk (47. ábra).

Vegyük észre, hogy a folyamatábrában szereplő két egymásba ágyazott ciklus FOR/NEXT ciklus képző utasításokkal is megoldható. Az indexeket, azaz egymás után következő természetes számokat, kell előállítanunk a ciklusokkal.

```
10  REM SORBARENDEZES
20  CLEAR 800:DIM A$(100)
30  CLS:I=1
40  REM BEOLVASAS
50  INPUT A$(I)
60  IF A$(I)<>"$"THEN I=I+1:GOTO 50
70  REM RENDEZES
80  N=I-1: FOR I=1 TO N-1
90  M=I
100 FOR J=I TO N
110 IF A$(J)<A$(M)THEN M=J
120 NEXT J
130 T$=A$(M):A$(M)=A$(I):A$(I)=T$
140 NEXT I
150 REM KIIRATAS
160 FOR I=1 TO N:PRINT A$(I):NEXT I
```

● Írjunk „kockadobó" programot! Számoljuk meg a különböző értékű dobásokat!

Hatféle dobás lehetséges, tehát hat változó szükséges a számláláshoz. Ezek a változók legyenek egy vektor elemei. Azt, hogy melyik változónak az értékét kell növelni, a dobás értéke adja meg. Ez lesz az index.

```
10  REM KOCKADOBAS
20  CLS:INPUT "KOCKADOBASOK SZAMA",N
30  RANDOM
40  FOR I=1 TO N
50  K=RND(6)
60  A(K)=A(K)+1
70  NEXT I
80  FOR I=1 TO6:PRINT I,A(I):NEXT I
```

● Írjunk programot, amivel az ezernél kisebb számokat betűkkel íratjuk ki!

Olyan programot már készítettünk, amellyel a számjegyek neveit írattuk ki. Ekkor a DATA adatlistából kerestük ki a számjegyekhez tartozó karakterláncot, a számjegyek nevét.

A szavakat most is a DATA adatlistában helyezzük el, de már nem onnan történik a kikeresés. A karakterláncokat beolvassuk egy vektorba, és az index segítségével választjuk ki a megfelelő elemet. Ez az eljárás nemcsak gyorsabb, mint a kikeresés, de a program szerkezete is egyszerűbb lesz.

```
 10  REM SZAMOK KIIRATASA BETUVEL
 20  DIM A$(22):INPUT "A SZAM <1000 ";X
 30  FOR I=1 TO 22:READA$(I):NEXT I
 40  X=1000+X
 50  FOR I=1 TO 3
 60  P(I)= X-10 * INT(X/10)
 70  X=INT(X/10)
 80  NEXT I
 90  IF P(3)=0THEN Z$=" " ELSE Z$="SZAZ"
100  IF P(2)=0 THEN 120
110  IF P(1)=0 AND P(2)<=2 THEN P(2)=P(2)+20 ELSE
     P(2)=P(2)+10
120  PRINT A$(P(3));Z$;A$(P(2));A$(P(1))
130  DATA EGY,KETTO,HAROM,NEGY,OT
140  DATA HAT,HET,NYOLC,KILENC, * * *
150  DATA TIZEN,HUSZON,HARMINC,NEGYVEN
160  DATA OTVEN,HATVAN,HETVEN,NYOLCVAN
170  DATA KILENCVEN,* * *,TIZ,HUSZ
```

A számjegyeket a P vektorba helyezzük el. Az X = 1000 + X utasítással megoldottuk azt, hogy mindig három számjegyű számmal dolgozhatunk. Nem kell vizsgálnunk a számjegyek számát.

Az egyesek és tízesek nevein kívül gondolni kell arra is, hogy a TIZ és HUSZ szavakra is szükségünk van, ha az egyesek helyén 0 áll és a tízesek száma 1 vagy 2.

Mivel az A$(0) változónak nem adtunk értéket, ezért 0 számjegy esetén is helyesen működik a programunk. Definiálatlan karakterlánc-változó értéke az „üres karakter".

● Készítsünk programot, amivel meghatározhatjuk egy N tagú brigád tagjainak heti, és a brigád napi teljesítményét!

A brigád tagjainak napi teljesítményét táblázatba foglaltuk:

Napok / Tagok	1	2	3	4	5	A tagok heti teljesítménye
1	23	47	18	28	37	
2	25	18	23	24	23	
.						
.						
A brigád napi tejesítménye						

A tagokat sorszámukkal azonosítjuk, ugyanúgy a napokat is. A táblázat egy adatát így *a sor és az oszlop sorszámával* egyértelműen megadhatjuk.

A táblázat adatait a programunkban *kétindexes változókban* tároljuk. Ezek a kétindexes változók egy mátrixnak lesznek az elemei. A vektort egy-, a mátrixot kétdimen-

ziós *tömbnek* is szokás nevezni. A táblázat adatai tehát: A(1,1), A(1,2), A(1,3) . . ., kétindexes változók értékei lesznek. Az első index a sort, a második az oszlopot mutatja.

Az adatok beírása több időt vesz igénybe, mint a program futtatása, és számolnunk kell a tévedéssel is, ezért az adatokat a DATA adatlistában helyezzük el. A könnyebb áttekinthetőség céljából egy sorba csak egy munkás teljesítményét írjuk.

A beolvasás soronként történik, amit úgy is mondunk, hogy a beolvasás *sorfolytonos.*

A munkások heti teljesítményét a B vektorba, a brigád napi teljesítményét a C vektorba összesítjük. A sorokat, ill. az oszlopokat összegezzük.

```
 10  REM  MUNKA KIERTEKELES
 20  CLS:INPUT "BRIGADLETSZAM",N
 30  DIM A(N,5),B(N),C(5)
 40  REM ADATBEOLVASAS
 50  FOR I=1 TO N:FOR J=1 TO 5
 60  READ A(I,J)
 70  NEXT J:NEXT I
 80  PRINT "TAGOK TELJESITMENYE"
 90  FOR I=1 TO N:FOR J=1 TO 5
100  B(I)=B(I)+A(I,J)
110  NEXT J:PRINTI,B(I):NEXT I
120  PRINT "NAPI TELJESITMENY"
130  FOR J=1 TO 5:FOR I=1 TO N
140  C(J)=C(J)+A(I,J)
150  NEXT I:PRINT J,C(J):NEXT J
500  REM ADATOK
510  DATA 23,47,18,28,37
520  DATA 25,18,23,24,23
```

● A brigád teljesítményét több héten keresztül értékeljük, és az adatokat mágnesszalagon tároljuk.

Állapítsuk meg a teljesítmények ismeretében, hogy a hét munkanapjain mennyire egyenletes a tagok és a brigád teljesítménye!

Ehhez a programhoz össze kell gyűjteni több hét teljesítményadatait, hogy ne kelljen újból begépelni azokat.

A programot kiegészítjük egy olyan programrészlettel, amivel az adatokat mágnesszalagra íratjuk ki. A kiírás a PRINT# -1, A(I, J) utasítással történik. A # (számjel) karakter utáni számmal a perifériát jelöljük, ahová kiírunk: 1 – beépített magnetofon, 2 – csatlakoztatott magnetofon.

```
200  REM ADATOK MAGNESSZALAGRA
210  PRINT "KAPCSOLD FELVETELRE A MAGNOT"
220  PRINT "ES NYOMJ LE EGY BILLENTYUT"
230  IF INKEY$="" THEN 230
240  FOR I=1 TO N:FOR J=1 TO 5
250  PRINT#-1,A(I,J)
260  NEXT J:NEXT I
```

Gondoljunk az *adatkivitel ellenőrzésére is.* Olvassuk be azokat még azzal a programmal, amelyben rendelkezésünkre állnak az adatok, és így hiba esetén megismételhetjük a kiíratást.

```
300  REM ELLENORZES
310  PRINT "CSEVELO VISSZA A SZALAGOT ES
     ALLITSD LEJATSZASRE A MAGNOTI";
     PRINT "NYOMJ LE EGY BILLENTYUTI"
320  ON  ERROR GOTO 400
330  IF INKEY$="" THEN 330
340  FOR I=1 TO N:FOR J=1 TO 5
350  INPUT#-1,X:PRINT X
360  IF INKEY$<>"" THEN ERROR 1
370   NEXT J:NEXT I
380  V$=INKEY$:IF V$="" THEN 380 ELSE IF
     V$="N" THEN 200 ELSE END
400  REM HIBAKEZELES
410  PRINT "KIIRATAS VAGY ELLENORZES (K-E)"
420  T$=INKEY$:IF T$="" THEN 420 ELSE IF
      T$="K" THEN RESUME 200 ELSE RESUME 300
```

A *hibafigyelést* a 320-as sorban az ON ERROR GOTO 400 utasítással indítjuk. A 350-es sorban az INPUT# −1, X utasítással olvassuk be az adatokat a szalagról, amiket azután a képernyőre is kiírunk, hogy ellenőrizhessük, mit olvasott be a program. Hibás adat esetén, ami adódhat a mágnesszalag hibájából, a 400-as utasítássorral folytatódik a program. Itt kezdődik a *hibakezelő programrészlet.* A beütött billentyűtől függően a programunk a 200-as, vagy a 300-as utasítássoron folytatódik, és az adatokat újból ellenőrizhetjük, vagy megismételhetjük a kiíratást.

A 360-as sorban *hibát szimulálunk* az ERROR 1 utasítással, ha lenyomunk beolvasás közben egy billentyűt. Előfordulhat, hogy a beolvasást nem elölről kezdtük, ilyenkor megszakíthatjuk a beolvasást, és beállíthatjuk újból a szalagot.

Készítsük el ezek után azt a programot, amivel a heti teljesítmények összehasonlítását végezzük!

A heti teljesítményadatokat egy *háromdimenziós tömbbe* olvassuk be, amelynek *elemei háromindexes változók* lesznek.

A harmadik indexszel a heteket jelöljük. Az A(3,4,2) indexes változó értéke ezek szerint azt jelenti, hogy a 3. munkás a 2. hét 4. napján mennyit teljesített.

A mágnesszalagról az adatokat az INPUT#−1, A(I,J,K) utasítással olvassuk be. A # karakter utáni számmal azt a perifériát jelöljük, ahonnan az adatokat beolvassuk.

A tagok teljesítményét a B mátrixban, a brigád teljesítményét a C vektorban összegezzük.

```
10  REM TELJESITMENY A HET NAPJAIN
20  INPUT "DOLGOZOK SZAMA",M
30  INPUT "HETEK SZAMA",N
40  DIM A(N,5,M),B(N,5),C(5)
```

```
 50  PRINT "ALLITSD A SZALAGOT AZ ADATOK ELEJERE
 60  PRINT "ES INDITSO EL A MAGNOT"  '
 70  FOR K=1 TO M:FOR I=1 TO N
 80  FOR J=1 TO 5
 90  INPUT#1,A(I,J,K)
100  NEXT J:NEXT I:NEXT K
110  REM OSSZEGZES
120  FOR J=1 TO 5 :FOR I=1 TO N
130  FOR K=1 TO M
140  B(I,J)=B(I,J)+A(I,J,K)
150  C(J)=C(J)+A(I,J,K)
160  NEXT K:NEXT I:NEXT J
170  REM KIIRAS
180  PRINT "TAGOK TELJESITMENYE"
190  FOR I=1 TO N:FOR J=1 TO 5
200  PRINT B(I,J),
210  NEXT J:PRINT:NEXT I
220  PRINT "BRIGAD TELJESITMENYE"
230  FOR I=1 TO 5:PRINT C(I):NEXT I
```

● Készítsünk programot a tíznél kisebb pozitív egész számok hatványainak kiszámítására!

A hatványok értékét a kétszeres pontosságú változókkal 16 számjegyig tudjuk meghatározni. Nagyobb számok meghatározása csak úgy lehetséges, ha a számjegyeket egy vektorban helyezzük el.

A MEM függvény a szabad memóriarekeszek számát adja. Egy egész változó 6 memóriarekeszt foglal el. Ezt felhasználva a DIM A(MEM/6) utasítással az A vektor számára lefoglaljuk a teljes szabad memóriaterületet.

A programmal azt az algoritmust követjük, ahogyan egy többszámjegyű számot egy egyjegyű számmal szorzunk. Az AT változó tartalmazza az átvitelt, a részszorzatok tízes értékét.

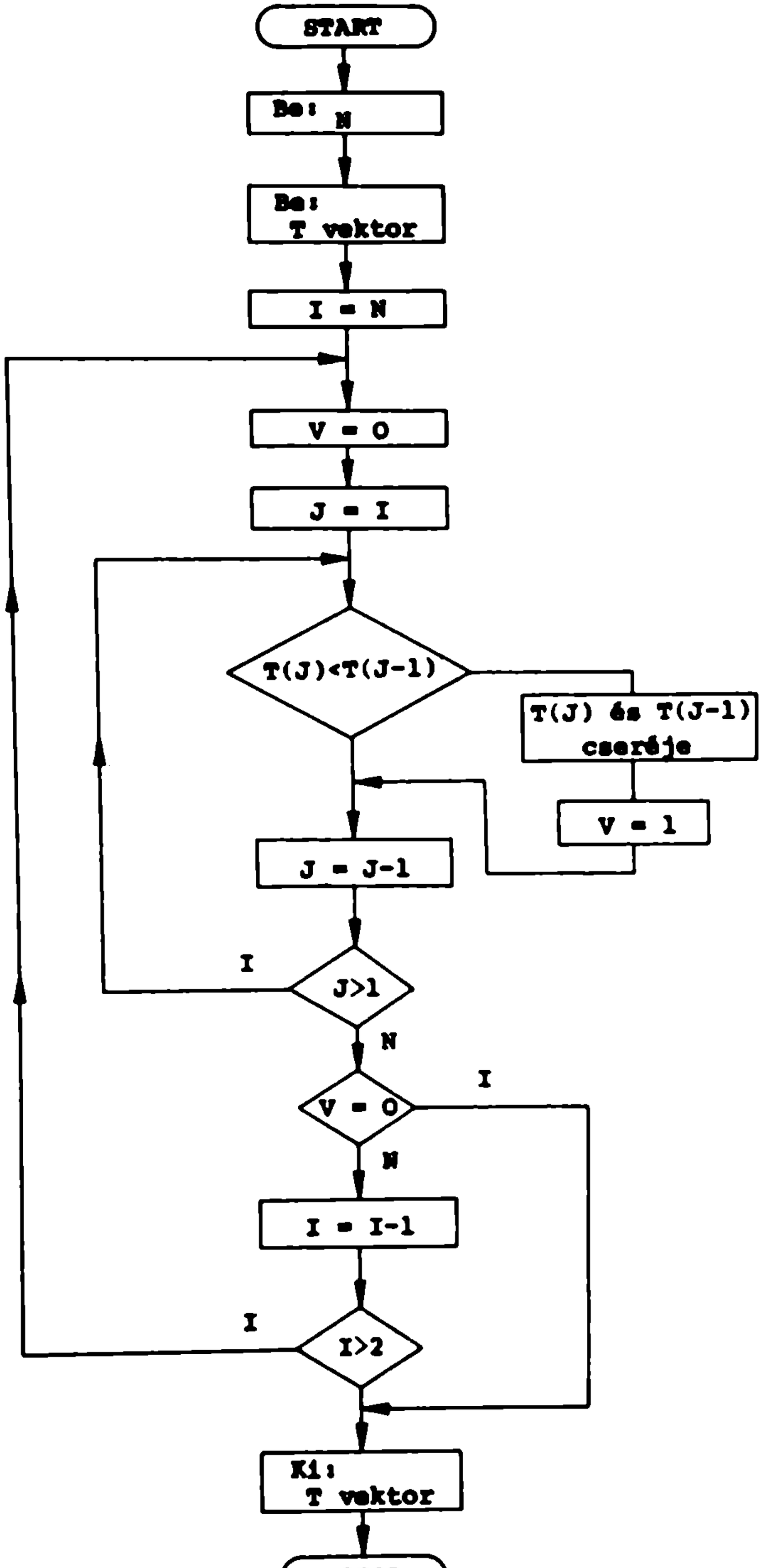
```
 10  REM HATVANYOK
 20  CLS:DEFINT A-Z:DIM A(MEM/6)
 30  INPUT "AZ ALAP <10",A
 40  N=1:REM A SZAMJEGYEK SZAMA
 50  K=0:REM A HATVANYKITEVO
 60  A(1)=1
 70  REM * * *SZORZAS * * *
 80  AT=0
 90  FOR I=1 TO N
100  A(I)=A(I) * A+AT
110  AT=A(I)/10:A(I)=A(I)-10 * AT
120  NEXT I
```

```
130  IF AT>0 THEN N=N+1:A(N)=AT
140  REM * * *KIIRATAS * * *
150  K=K+1:PRINT A;"↑";K;"=",
160  IF N=1 THEN 180
170  FOR L=N TO 2 STEP-1:PRINT A(L) ,:NEXT L
180  PRINT A(1)
190  IF N<MEM/6 THEN 70
```

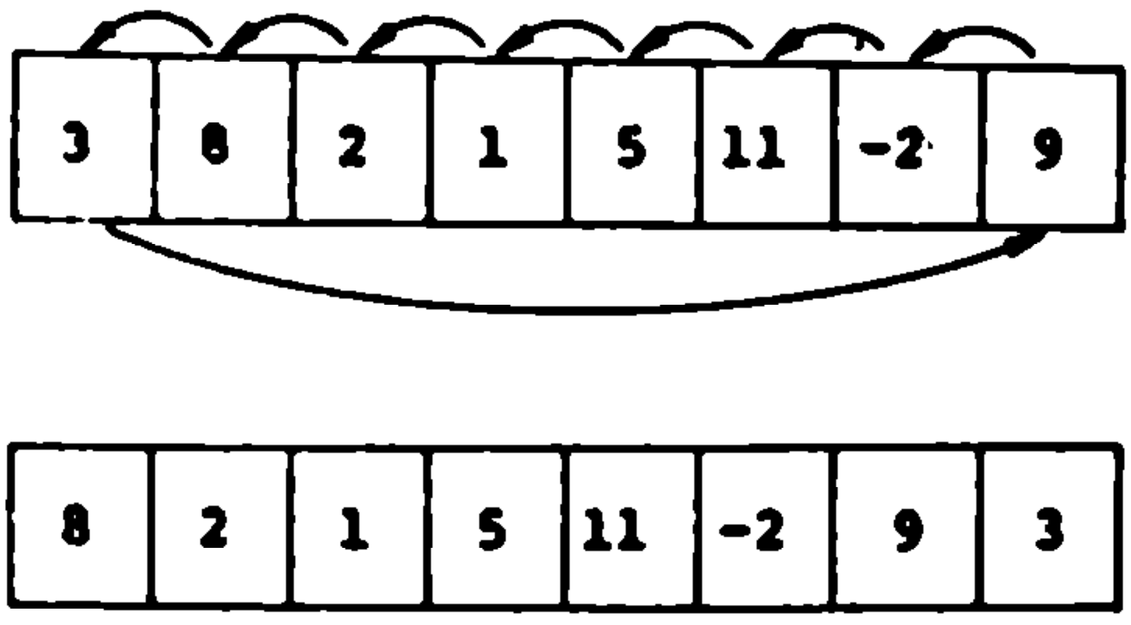
Az alap az A változó értéke lesz. Megkülönböztetésül, a nem indexes változókat *egyszerű változóknak* nevezzük. Egy programban ugyanazzal a névvel jelölhetünk egyszerű változót és különböző dimenziójú tömböket is.

Kérdések és feladatok

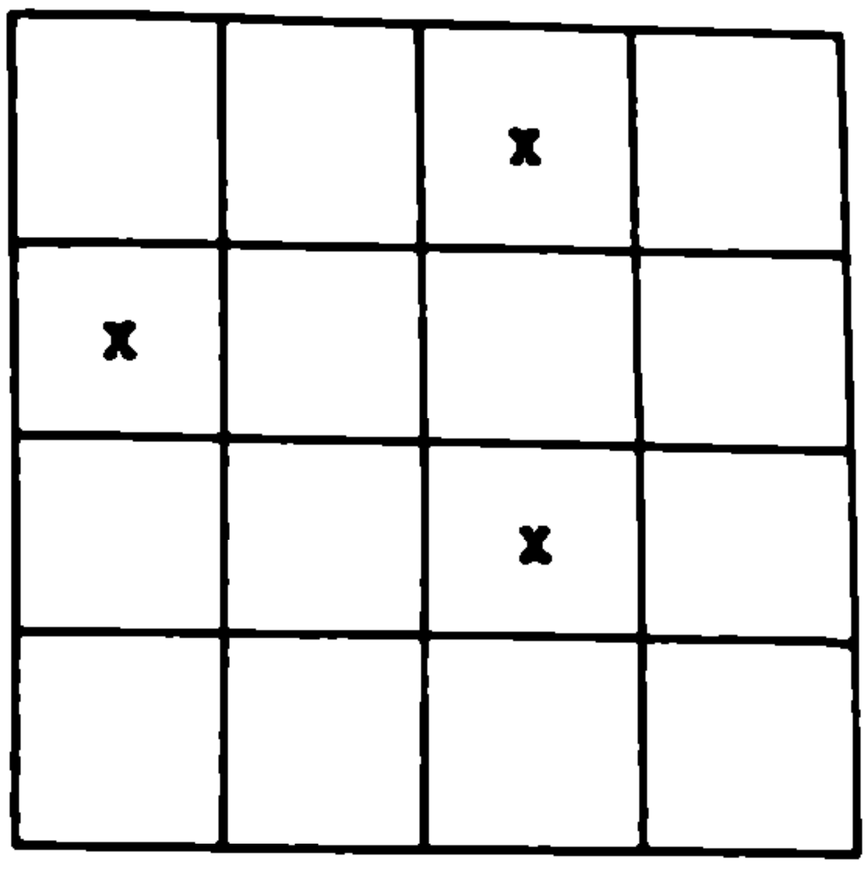
1. Hogyan csoportosíthatjuk a változókat?
2. Hogyan nevezzük az azonos nevű egy-, ill. kétindexes változókat?
3. Mi a DIM utasítás szerepe?
4. Mikor kell használni a CLEAR utasítást?
5. Rendezzük növekvő sorrendbe a T vektor elemeit az ún. „buborék" módszerrel (48. ábra)! (N > 2). Kövessük végig az algoritmust konkrét adatokkal!
6. Készítsünk programot, amivel megszámlálhatjuk az azonos osztályzatú dolgozatok számát, és kiszámíthatjuk az osztályátlagot!
7. Használjuk fel a totótippeket adó programunkhoz a T$(1) = "X":T$(2) = "1": :T$(3) = "2" értékadó utasításokat!
8. Határozzuk meg a prímszámokat az eratoszthéneszi-szita módszerével!
9. Írjunk programot, amivel egy vektor elemeit egy hellyel körbeforgathatjuk (49. ábra)!
10. Egészítsük ki a SZAMOK KIIRASA BETUVEL programot úgy, hogy 1000-nél nagyobb számokat is lehessen betűvel kiíratni!
11. Írjunk programot, amivel kiszámíthatjuk egy osztály tanulóinak és a tantárgyaknak az átlagát!
12. Kíséreljük meg „lóugrással" bejárni a sakktáblát! Csak oda léphetünk, ahol még nem voltunk! A lépések lehetnek véletlenszerűek, de követhetünk valamely szabályt is!
13. Készítsünk „torpedó" játékot! Véletlenszerűen elhelyezett „hajók" helyzetét kell eltalálnunk (50. ábra)!
14. Írjunk lottószámokat előállító programot! A számokat növekvő sorrendben írassuk ki, és gondoskodjunk arról, hogy az öt szám között ne szerepeljen két egyforma!
15. Írjunk programot két 20 számjegyű, tetszőleges számrendszerbeli szám összegzésére!
16. Írjunk programot, amivel szimulálhatunk egy tetszőleges számrendszerben számoló kilométerórát!
17. Készítsünk „rakéta" játékot! Véletlenszerűen elhelyezett „repülők" helyzetét kell eltalálnunk!


START
Be: N
Be: T vektor
I = N
V = 0
J = I
T(J)<T(J-1)
T(J) és T(J-1) cseréje
V = 1
J = J-1
I
J>1
N
I
V = 0
N
I = I-1
I
I>2
Ki: T vektor
STOP

48. ábra



49. ábra



50. ábra

KARAKTERLÁNCOK

A BASIC nyelvben az *adatokat* két csoportba soroljuk: számok és karakterláncok. A számokat numerikus adatoknak, a karakterláncokat stringeknek is nevezik.

Numerikus adatok kezelésével, az azokkal végezhető műveletekkel az aritmetikai kifejezések és az aritmetikai függvények tárgyalásakor már megismerkedtünk.

A *karakterláncok* olyan karaktersorozatok, amiket a képernyőn megjeleníthetünk: betűk, számjegyek, írásjelek. Az ilyen típusú adatokat *karakterlánc-változókban* tároljuk, amiket a DEFSTR utasítással jelölünk ki vagy a $-jellel különböztetjük meg a numerikus változóktól.

- Értelmezzük a következő programot!

```
10  REM KARAKTERLANCOK -1
20  A$="METER"
30  INPUT "MEROSZAM",M
40  INPUT "ELOTAG",E$
50  PRINT M,E$,A$
```

A mértékegység megfelelő előtagja pl. KILO, DECI vagy MIKRO, az E$, a mértékegység az A$ karakterlánc-változó értéke lesz. A hosszúságot így egy adott egységben írathatjuk ki.

A következő program ugyanezt a feladatot oldja meg, csak a karakterlánc-változókat most a DEFSTR utasítással adjuk meg.

```
10  REM KARAKTERLANCOK -2
20  DEFSTR A,E
30  A="METER"
40  INPUT "MEROSZAM",M
50  INPUT "ELOTAG",E
60  PRINT M,E,A
```

Az értékadó utasításban a karakterláncot idézőjelek közé tesszük. Idézőjelek közé írjuk a PRINT vagy az INPUT utasítással kiírandó karakterláncot is.

A karakterláncokkal műveletet is végezhetünk: kettő vagy több karakterláncot *összefűzhetünk* eggyé. A hosszúság egységét az első programunkban a B$ változóba írhatjuk a következő értékadó utasítással: B$ = E$ + A$.

Egy karakterláncban legfeljebb 255 karakter lehet. Hosszabb karakterláncok esetén azonban már gondolnunk kell arra, hogy rendelkezésünkre álljon a tárolásához szükséges memóriaterület, amit a CLEAR utasítással biztosíthatunk. Egy karakter egy memóriarekeszt foglal le.

● Írjunk programot, ami tízes számrendszerben adott számokat kettes számrendszerbe ír át!

Ezt a programot indexes változókkal egyszer már megoldottuk. Az algoritmust ott írtuk le. Most a karakterláncok összefűzésével oldjuk meg azt, hogy a bináris számjegyek megfelelő sorrendben legyenek.

```
10  REM KONVERTALAS KETTES SZAMRENDSZERBE
20  INPUT "A SZAM";X
30  I=0:X$=""
40  H=INT(X/2):B=X-2*H
50  IF B=1 THEN B$="1" ELSE B$="0"
60  X$=B$+X$
70  X=H:IF X<>0 THEN 40
80  PRINT X$
```

Az X$ változóba – amelyik kezdetben „üres" volt – írjuk a biteket. A legkisebb helyiértékű bitet kapjuk meg először, ezért a következőket a már meglevő számjegyek elé helyezzük (60-as sor).

Az 50-es utasítássorban a *B értékéből karakterláncot állítunk elő,* hogy az „összefűzést" el tudjuk végezni.

Ezt valósítja meg a STR$ karakterlánc függvény is. Ekkor az utasítássor:

```
50  B$=STR$(B)
```

Ez a függvény a B változó numerikus értékét karakterlánccá alakítja. Argumentumként kifejezés is szerepelhet, ezért a következő utasítássorok is ugyanazt az eredményt adják:

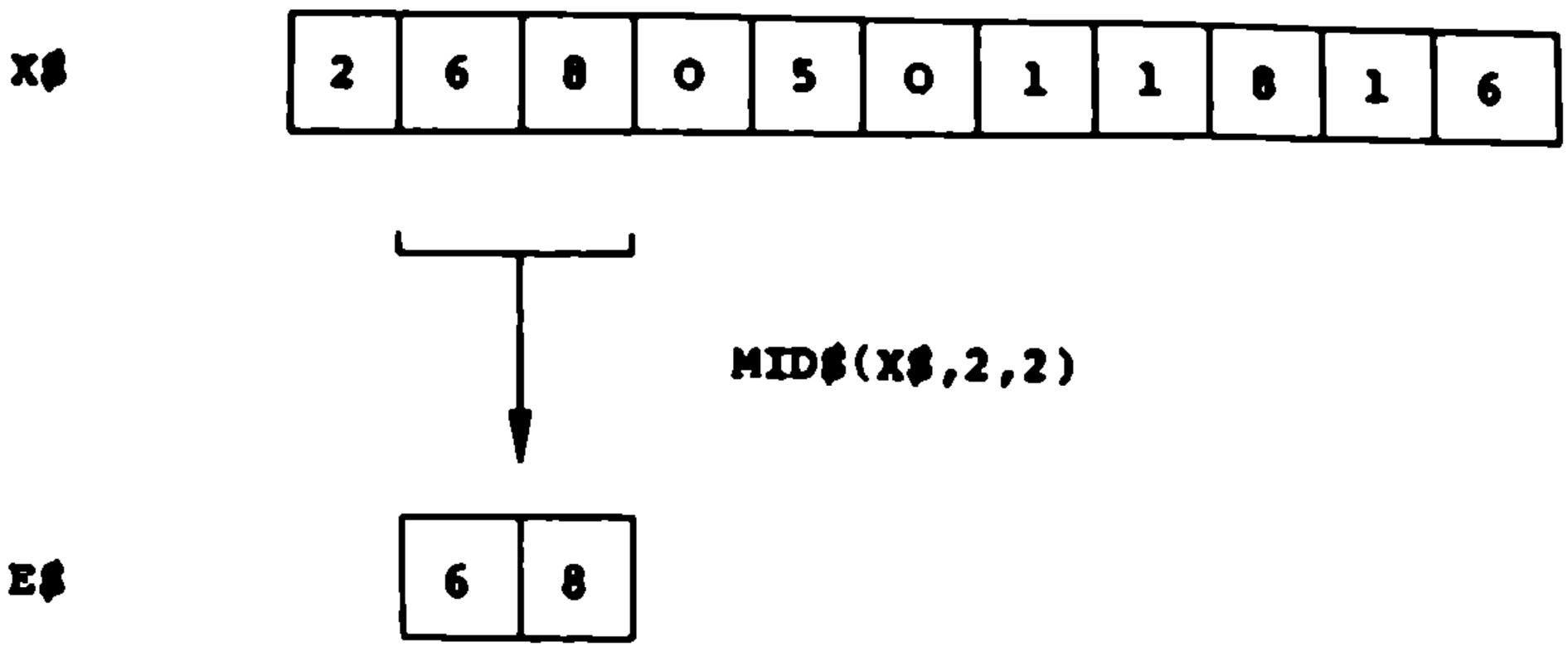
```
40  H=INT(X/2)
50  B$=STR$(X-2*H)
```

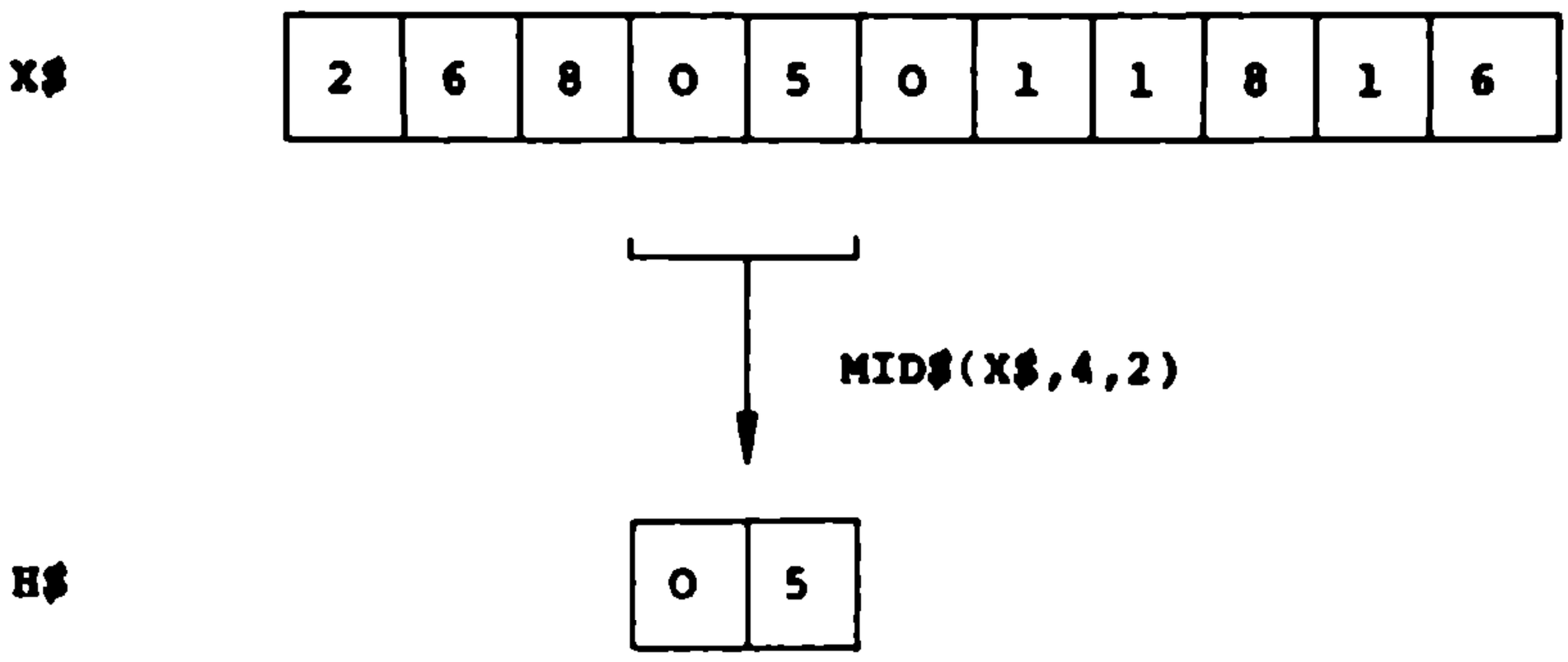
A STR$ függvény inverze a VAL függvény, amivel *numerikus karakterláncot* tudunk *numerikus adattá* alakítani.

● Állapítsuk meg program segítségével, hogy hány éves és hány hónapos az illető, ha ismerjük a személyi számát!

A személyi szám 2. és 3. karaktere a születési év utolsó két számjegye. Ezt a karaktert a MID$ *függvénnyel „vesszük ki"* a személyi számból, és átalakítjuk numerikus adattá a VAL függvénnyel, hogy számolni tudjunk vele.

A MID$ *függvény argumentumában* azt kell megadnunk, hogy *melyik karakterláncból,* és a karakterlánc *hányadik karakterétől* kezdődően, *mennyi karaktert* „veszünk ki". A „kivételt" úgy kell értelmeznünk, hogy az argumentumban szereplő karakterlánc változatlan marad (51. ábra).





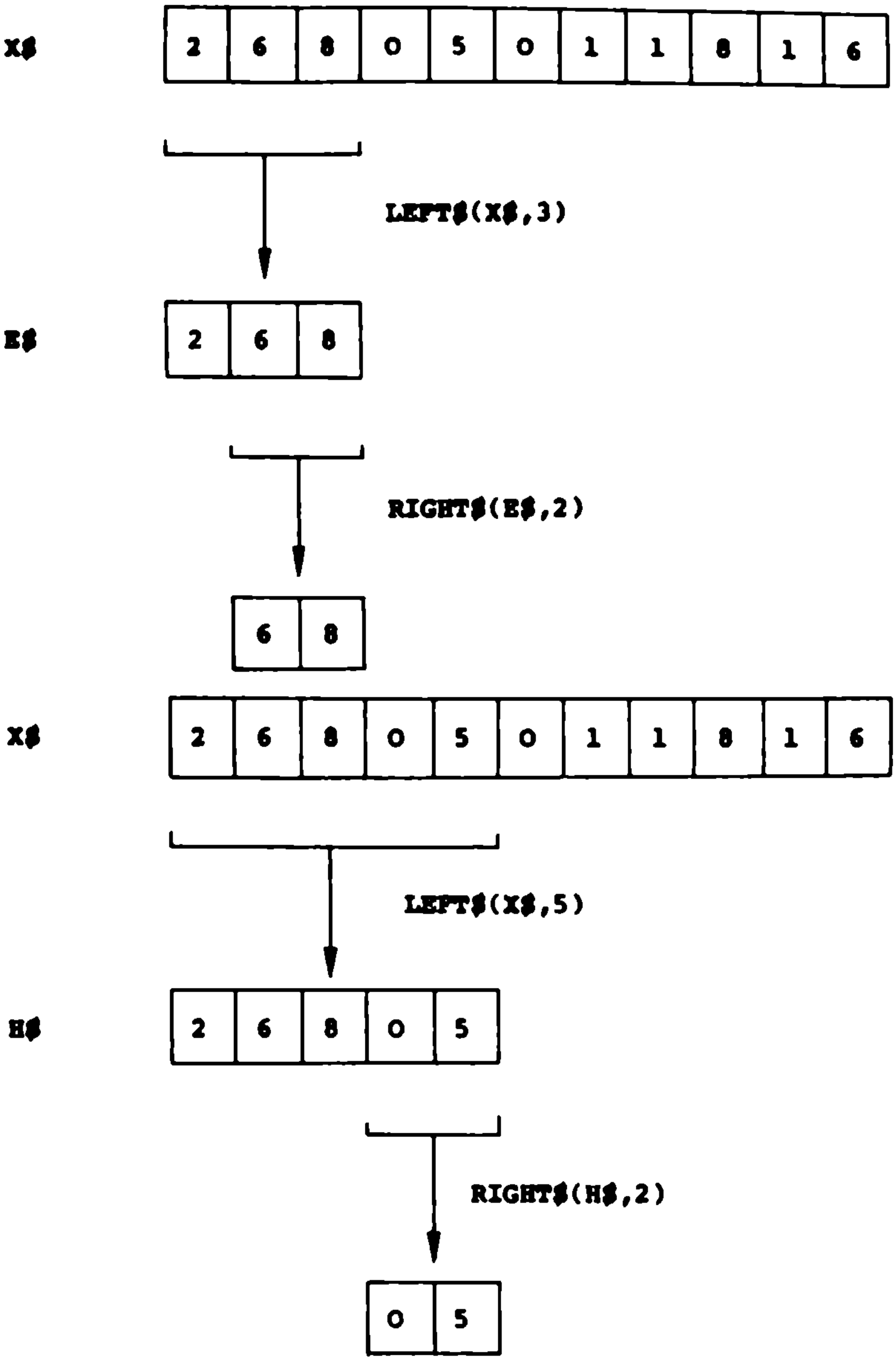
51. ábra

Az E$ változóba a születési év utolsó két számjegyét, a H$ változóba a születés hónapját helyezzük el.

```
10  CLS:REM ELETKOR
20  INPUT "FOLYO EV ES HONAP",A,B
30  INPUT "SZEMELYISZAM";X$
40  F=A-1900
50  E$=MID$(X$,2,2):H$=MID$(X$,4,2)
60  E=VAL(E$):H=VAL(H$)
70  E=F-E:H=B-H
80  IF E<=0 AND H<0 THEN PRINT "EZ NEM LEHET
                                    :GOTO 20
90  IF H<0 THEN E=E-1:H=12+H
100 PRINT "A";X$;"SZEMELYISZAM TULAJDONOSA"
110 PRINT E, "EVES ES";H,"HONAPOS"
```

A „kivételt" megoldhatjuk a LEFT$ és a RIGHT$ *függvényekkel* is. Ezekkel a függvényekkel *egy karakterlánc elejéről, ill. végéről választhatunk le adott számú karaktert* (52. ábra).

A függvények argumentumában függvények is szerepelhetnek, tehát az 50-es és a 60-as utasítássorok helyett írhatjuk a következőket is:



52. ábra

```
50  E=VAL(MID$(X$,2,2,))
60  H=VAL(MID$(X$,4,2))   , vagy

50  E=VAL(RIGHT$(LEFT$(X$,3),2))
60  H$=RIGHT$(LEFT$(X$,5),2):H=VAL(H$)   , vagy

50  Y$=MID$(X$,2,4)
60  E=VAL(LEFT$(Y$,2)):H=VAL(RIGHT$(Y$,2))
```

• Írjunk programot, amivel meghatározhatjuk, hogy egy karakterláncban egy-egy betű hányszor fordul elő!

```
10  REM BETUK ELOFORDULASA
20  DIM T(26)
30  INPUT "SZOVEG",X$
40  FOR I=1 TO LEN (X$)
50  B$=MID$(X$,I,1)
60  L=ASC(B$)-64
70  T(L)=T(L)+1
80  NEXT I
90  REM KIIRATAS
100 FOR I=1 TO 26
110 PRINT CHR$(64+I),"/",T(I)," - ",
120 NEXT I
```

Az ASC *függvénnyel előállítjuk a karakterlánc karaktereinek kódját*, amiből kiszámíthatjuk, hogy a karakter az abc-ben hányadik helyen áll. Ezt mutatja az L változó értéke. A T vektor L-edik elemének az értékét 1-gyel növelve, a T vektorban megkapjuk, hogy az abc betűi hányszor fordulnak elő a karakterláncban.

A LEN *függvénnyel* egy *karakterlánc karaktereinek számát* határozhatjuk meg, azaz egy karakterlánc hosszát. A LEN(X$) függvény értéke megadja, hogy hány karaktert kell kiválasztanunk a vizsgálathoz az X$ karakterláncból. A karakterek kiválasztását a MID$ függvénnyel végezzük.

A betűk előfordulásának száma mellé a CHR$ függvénnyel kiíratjuk az abc betűit is. A CHR$ *függvény argumentumában* szereplő 64 + I kifejezés értékével az abc *betűinek kódját* állítjuk elő az I értékének változtatásával.

Az ASC függvény a karakterlánc első karakterének kódját adja meg, ezért az 50-es utasítássor helyett írhatjuk az

```
50  B$=RIGHT$(X$,LEN(X$)-I+1)
```

utasítássort is.

• Írjunk programot, amivel kiírathatjuk a karakterek kódját és a karaktereket!

```
10  REM KARAKTERKESZLET
20  FOR I=0 TO 255
30  PRINT I," - ",CHR$(I)
40  NEXT I
```

A karaktereket kódszámaik szerint a következő csoportokra oszthatjuk!

0– 31	– vezérlő karakterek,
32–127	– a billentyűzet karakterei,
128–191	– grafikus karakterek,
192–255	– szóköz karakterek.

A *vezérlő karakterek* ún. nem megjelenő karakterek. Kiírásukkal a képernyő állapotát, a kiíratás módját határozhatjuk meg. Az egyes karakterek hatásának leírását a 9. mellékletben találjuk. A karaktereket olyan gyorsan írja ki a program, hogy nem tudjuk követni.

Futtassuk lépésenként a programot! Írjuk be a következő utasítássort:

```
35  IF INKEY$=""  THEN 35
```

Az INKEY$ függvény értéke az a karakter lesz, amit a függvény hívásakor begépelünk. Amennyiben nem nyomunk le billentyűt, az INKEY$ függvény értéke az „üres karakter", amit a programban két egymás mellé írt idézőjellel jelzünk.

A 35-ös soron a program tehát mindaddig várakozik, ameddig le nem nyomunk egy billentyűt.

A SHIFT és @billentyűk egyszerre történő lenyomásával is megszakítható a program futása. A program végrehajtása ebben az esetben bármelyik billentyű lenyomására folytatódik.

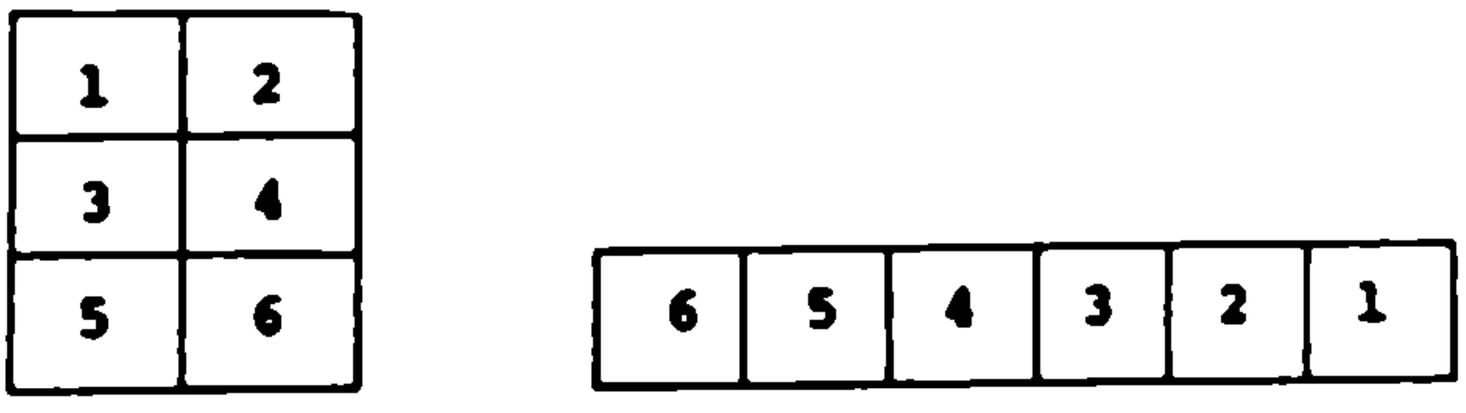
- Adjuk meg úgy egy karakterlánc-változó értékét, hogy a karakterek ne jelenjenek meg a képernyőn!

Az INKEY$ függvénnyel beolvasott karaktereket összefűzzük. Ezek a karakterek nem jelennek meg a képernyőn. A NEW LINE billentyű lenyomásával jelezzük az adatbegépelés befejezését. A NEW LINE karakter kódja 13.

```
10  REM LATHATATLAN BEIRAS
20  X$=""
30  A$=INKEY$:IF A$="" THEN 30
40  IF A$=CHR$(13)THEN PRINT X$:END
50  X$=X$+A$:GOTO 30
```

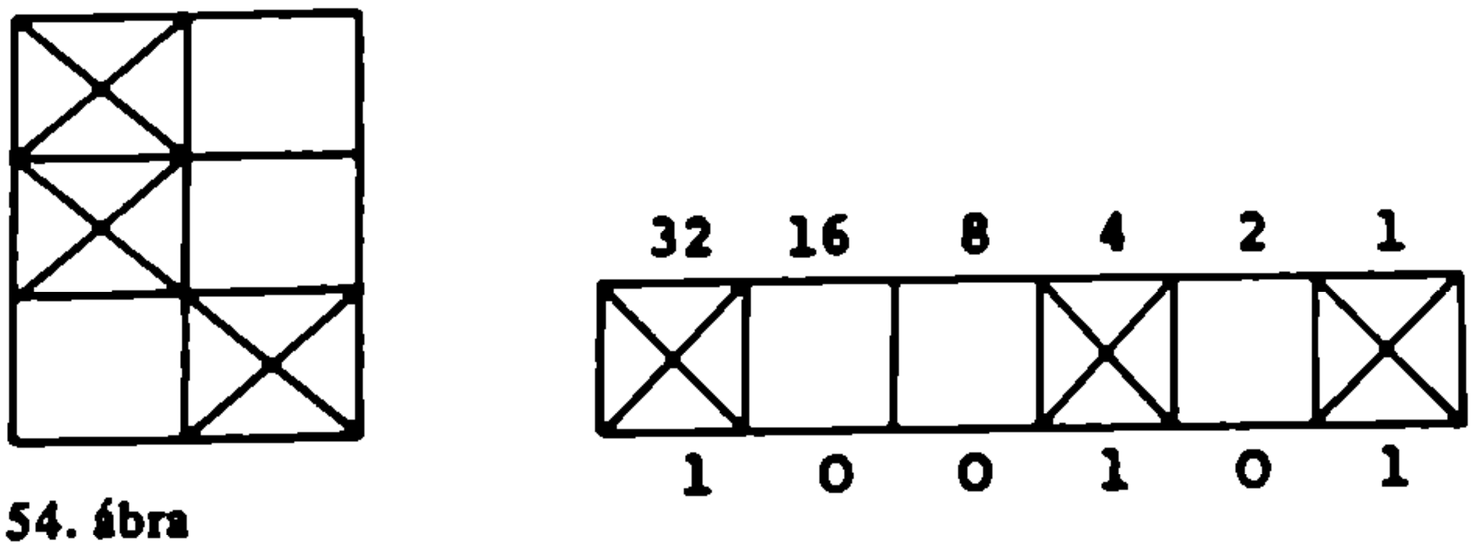
- Jelenítsük meg a grafikus karaktereket! A képernyőn 3x2-es pontmátrixszal adhatjuk meg a grafikus karaktereket. Hat pontra beosztható egy karakterpozíció.

Sorszámozzuk a pontokat, és helyezzük el a sorszámoknak megfelelően egy sorban (53. ábra).



53. ábra

Tekintsük a következő grafikus karaktert (54. ábra): A világító pontokat jelöljük 1-gyel, ill. 0-val. Kapunk egy bináris számot, amelyiknek decimális értéke 37.



54. ábra

A grafikus karakterek közül a legkisebb kódszámú a 128-as. Ennek egyik pontja sem világít, tehát a pontjaiból alkotott szám 0.

A keresett grafikus karakter kódszámát úgy kapjuk meg, hogy a 128-hoz hozzáadjuk a 37-et. A PRINT CHR$(165) utasítással kiírathatjuk a keresett karaktert.

A következő program az összes grafikus karaktert előállítja:

```
10  REM GRAFIKUS KARAKTEREK
20  FOR I=0 TO 63
30  PRINT I,CHR$(128+I),
40  NEXTI
```

Amikor minden pont világít, a pontmátrixból alkotott bináris szám értéke 63. Tehát az utolsó grafikus karakter kódja 191.

- Világítsuk ki a képernyő minden pontját!

A STRING$ *függvénnyel* azonos karakterekből álló karakterláncot tudunk előállítani. Argumentumként meg kell adni a karakterlánc hosszát és a karakterek kódját.

A képernyő utolsó karakterpozíciójába nem írhatunk karaktert, mert akkor automatikus soremelés történik, ezért ennek a karakterpozíciónak a pontjait a SET utasítással világítjuk ki.

```
10  REM VILAGOS KEPERNYO
20  CLS:CLEAR 100
30  FOR I=1 TO 15
40  PRINT STRING$(64,191),
50  NEXT I
60  PRINT STRING$(63,191),
70  FOR J=1 TO 3:FOR I=1 TO 2
80  SET(125+I,44+J)
90  NEXT I:NEXT J
100 IF INKEY$="" THEN 100
```

Ha pl. X betűvel szeretnénk teleírni a képernyőt (az utolsó karakterpozíció kivételével), akkor a STRING$ (64, 88) vagy STRING$ (64, "X") utasításokat kellene használnunk. Ebben az esetben az argumentumban a karakter kódja helyett írhatjuk magát a karaktert.

- Csillag karakterekből rajzoljunk ki a képernyőre egy téglalapot!

```
10  REM TEGLALAP
20  INPUT "OLDALAK",N,M:CLS
30  PRINT STRING$(M+2,"*")
40  FOR I=1 TO N
50  PRINT "*",CHR$(192+M),"*"
60  NEXT I
70  PRINT STRING$(M+2,"*");
```

A CHR$ (192 + M) kiírásával M darab szóköz karaktert állítunk elő.

- Véletlenszerűen imbolyog egy tengerész a hídon? Grafikus karakterekből rajzoljuk meg a hidat és a tengerészt!

```
10  REM IMBOLYGO TENGERESZ
20  CLS
30  FOR I=0 TO 63:PRINT@9*64+I,CHR$(191),:NEXT
40  RANDOM:P=32
50  Q=P+SGN(RND(0)-0.5)
60  IF Q<0 OR Q>61 THEN PRINT@11*64,"LEESETT":END

70  PRINT@7*64,TAB(Q),CHR$(178),CHR$(177),
80  PRINT@8*64,TAB(Q),CHR$(150),CHR$(169),
90  PRINT@7*64,TAB(P)," "
100 PRINT@8*64,TAB(P)," ",
110 P=Q:GOTO 50
```

● Írjuk át a 64-nél kisebb egész számokat kettes számrendszerbe!

Ha az adott szám X, akkor a 128 + X kódszámú grafikus karakter pontjai X bináris értékét adják. A pontok állapotát megvizsgálva, meghatározhatjuk a szám kettes számrendszerbeli alakját.

```
10  REM ATALAKITAS!!
20  CLS:INPUT "A SZAM<64",X
30  PRINT@64,CHR$(128+X):X$=""
40  FOR I=3 TO 5:FOR J=0 TO 1
50  IF POINT(I,J)THEN X$="1"+X$ ELSE X$="0"+X$
60  NEXT:NEXT
70  PRINT X$
```

Kérdések és feladatok

1. Hogyan kell megadni egy karakterláncot
a) az értékadó utasításban,
b) az INPUT utasításnál,
c) a DATA adatlistában?
2. Hogyan különböztetjük meg a numerikus változóktól a karakterlánc-változókat?
3. Milyen hosszú lehet egy karakterlánc?
4. Mi a CLEAR utasítás jelentése, és mikor használjuk?
5. Csoportosítsuk a karakterlánc-függvényeket!
   – a karakterláncról ad információt,
   – a karakterláncból új karakterláncot állít elő,
   – numerikus karakterláncokkal kapcsolatosak,
   – a karakterek és kódjaik között teremtenek kapcsolatot.
6. Hogyan lehet eldönteni egy helyesen felírt függvényről, hogy a függvényérték karakterlánc vagy numerikus adat lesz-e?
7. Milyen karakterek szerepelhetnek az STR$ függvényértékeként kapott numerikus karakterláncban a számjegyeken kívül?

8. Írjunk programot, amivel egy mondatot szavakra bonthatunk!

9. Keressük ki a háromjegyű számok közül azokat, amelyek egyenlőek számjegyeik köbeinek összegével!

10. Írjunk programokat, amikkel az oszthatóságot vizsgáljuk, az oszthatósági szabályok alkalmazásával!

11. Írjunk programot annak eldöntésére, hogy egy karakterláncban megtalálható-e egy másik karakterlánc?

12. Rajzoljunk a képernyőre grafikus karakterek kiíratásával vízszintes és függőleges egyeneseket!

13. Mire használhatjuk az

```
40 IF INKEY$="" THEN 40
```

utasítást?

14. Grafikus karakterek kiírásával jelenítsük meg a képernyő minden második grafikus pontját!

15. Állítsuk abc sorrendbe egy szó betűit!

16. Írjunk programot, amivel meghatározhatjuk, hogy egy betű hányszor szerepel egy adott szóban!

17. Rajzoljunk sakktáblát a képernyőre!

SZUBRUTIN

• Írjunk függvénytáblázatot készítő programot! Írassuk ki a trigonometrikus függvények értékeit 0,05-os lépésközönként a $0 \leqslant x < \pi$ intervallumban!

Feltételek az output-ra:

– Készítsünk fejlécet, amelyben kiírjuk a függvények neveit!

– Az X értékét és a hozzá tartozó függvényértékeket egy sorban jelentessük meg. A képernyőn egyszerre legfeljebb tíz sor legyen látható.

– A függvényértéket négy tizedesjegy pontossággal írassuk ki! A negyedik tízes legyen kerekített érték.

A program folyamatábrájában (55. ábra) két helyen is szerepel a fejléc kiírása. Először a ciklusba lépés előtt, majd a cikluson belül, minden tizedik sor kiírása után.

A fejléc kiírása a programon belül egy részfeladat. Ezt egy szubrutinnal oldjuk meg.

A *szubrutin* olyan részprogram, amelynek végrehajtása a GOSUB utasítás hatására kezdődik, és egy RETURN utasítás végrehajtásával fejeződik be.

```
 10  REM FUGGVENYTABLAZAT
 20  GOSUB 100:REM FEJLEC
 30  X=0:I=1
 40  GOSUB 140:REM KIIRATAS
 50  X=X+0.05:X=INT(100*X+0.5)/100:I=I+1
 60  IF I<=10 THEN 80
 70  IF INKEY$="" THEN 70 ELSE GOSUB 100:I=1
 80  IF X<4*ATN(1)THEN 40 ELSE END
100  REM FEJLEC
110  CLS:PRINT TAB(28)"FUGGVENYTABLAZAT":PRINT
```

```
120   PRINT TAB(15),"SIN(X)",TAB(28),
      "COS(X)",TAB(41),"TAN(X)",TAB(53),
      "COTAN(X)"
130   RETURN
140   REM FUGGVENYERTEKEK
150   PRINT "X=",X,TAB(8),
160   Y=SIN(X):GOSUB 210
170   Y=COS(X):GOSUB 210
180   IF X=2*ATN(1)THEN PRINT STRING$
      (7,32)+"--",ELSE Y=TAN(X):GOSUB 210
190   Y=TAN(X):IF Y<>0 THEN Y=1/Y:GOSUB 210:
      PRINT ELSE PRINT STRING$(7,32)+"--"
200   RETURN
210   REM KIIRATAS-PRINTUSING
220   F$=STRING$(10-LEN(STR$(INT(ABS(Y)))),35)+
      ".### ":PRINTUSINGF$,Y,
230   RETURN
```

A fejléckiíró szubrutinunk a 140-es utasítássorral kezdõdik, ezért a GOSUB 140 utasítással „hívhatjuk" a szubrutint. A szubrutin hívó utasítás hatására a 140-es utasítássoron folytatódik a program. A szubrutinban maradunk mindaddig, ameddig egy RETURN utasítás végrehajtására nem kerül sor.

A RETURN utasítás után a program futása a szubrutin hívó utasítást követõ utasítással folytatódik.

Összetettebb feladatokat részfeladatokra bonthatunk, és megoldásukra szubrutinokat írhatunk. A programnak azt a részét, ahonnan ezeket a részfeladatokat megoldó szubrutinokat hívjuk, *fõprogramnak* nevezzük.

Mintaprogramunkban a fõprogramból még egy szubrutint hívunk. Ez a szubrutin a függvényértékeket számítja ki, és a megadott formátumban kiírja.

A kiíratást is egy szubrutin végzi. Ennek kezdõ sorszáma 210. Hívása a szubrutinból történik. Szubrutinok egymásba skatulyázásának nevezzük azt, amikor szubrutinból hívunk egy másik szubrutint. A RETURN utasítás hatására mindig az utoljára végrehajtott GOSUB utasításhoz tér vissza a program, és az azt követõ utasítás végrehajtásával folytatódik.

A *szubrutin hívó utasítás* alakja:

GOSUB sorszám

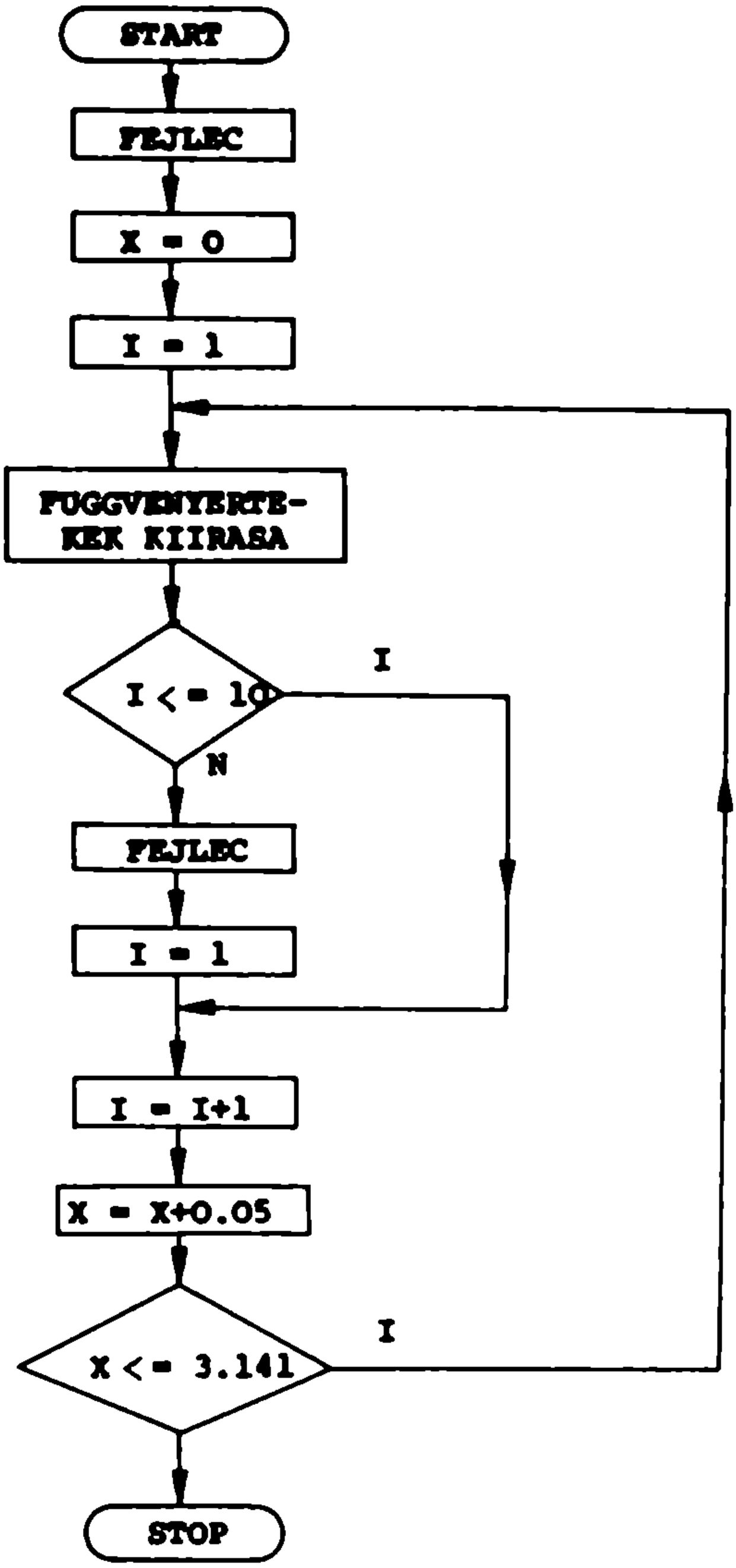
jelentése: „Menj az adott sorszámmal kezdõdõ szubrutinra!"

hatása: a GOSUB utasításban szereplõ sorszámmal folytatódik a program egy RETURN utasítás végrehajtásáig.

Visszatérés a szubrutinból:

RETURN

jelentése: „Térj vissza az utoljára végrehajtott GOSUB utasításhoz és a GOSUB utasítást követõ utasítást hajtsd végre!



55. ábra

• Oldjuk meg a függvényértékek adott formátumú kiíratását a PRINTUSING utasítás alkalmazása nélkül!

A szubrutinunk a következő:

```
210  REM KIIRATAS
215  Y9$=STR$(YX10 ↑ 9):A9=
     LEN(Y9)-2:IF MID(Y9$,A9,1)="+"
     THEN PRINT Y;:RETURN
220  Y=INT(YX10↑T9):Q$=STR$(ABS(Y))
230  Q$=STRING$(10,"0")+RIGHT$(Q$,LEN(Q$)-1)
```

```
240   T$=RIGHT$(Q$,T9):E$=STR$(VAL(LEFT$(Q$,
      LEN(Q$)-T9)))
250   E$=RIGHT$(E$,LEN(E$)-1):IFY<0 THEN E$=
      "-"+E$
260   E$=STRING$(T9-LEN(E$),32)+E$
270   PRINT E$;".";T$
280   RETURN
```

Kész szubrutinok alkalmazásához nem elegendő az, hogy ismerjük a kezdő sorszámot. Meg kell adni, hogy milyen változókat használ a szubrutin, és azok közül melyeknek kell értéket adnunk, mielőtt kérnénk a szubrutint. Ezek lesznek *a szubrutin bemenő paraméterei*, amiket a főprogramban adunk meg.

A KIIRATAS szubrutin bemenő paraméterei:

T9 – kiírandó tizedesjegyek száma

E9 – egész számjegyek és a számjegy előtti szóközkarakterek számának összege

Y – függvényérték

változói: Q$, E$, T$

kimenő paramétere nincs.

Alkalmazása:

– egyszeres pontosságú számok kiírására alkalmas;

– CLEAR 200 utasítás szükséges.

A szubrutin által használt változók ismerete azért szükséges, hogy ilyeneket a főprogramban ne használjunk. Ezért jelöltük ezeket ritkán használt nevekkel.

A szubrutinnak nincs kimenő paramétere, azaz nem kell értéket átadnunk a főprogramnak. Y értéke, ami a KIIRATAS szubrutin bemenő paramétere, az a FUGGVENYERTEKEK szubrutinnak kimenő paramétere lesz. Itt történik az értékadás, és ezzel dolgozik a másik szubrutin.

A szubrutin alkalmazása esetén a szubrutinon kívül a következő utasítássort is be kell írnunk a programba:

```
15  CLEAR 200:T9=Y:E9=8
```

Az eddigiek alapján a szubrutinok alkalmazása a következő előnyökkel jár:

– összetett feladatot részfeladatokra lehet bontani;

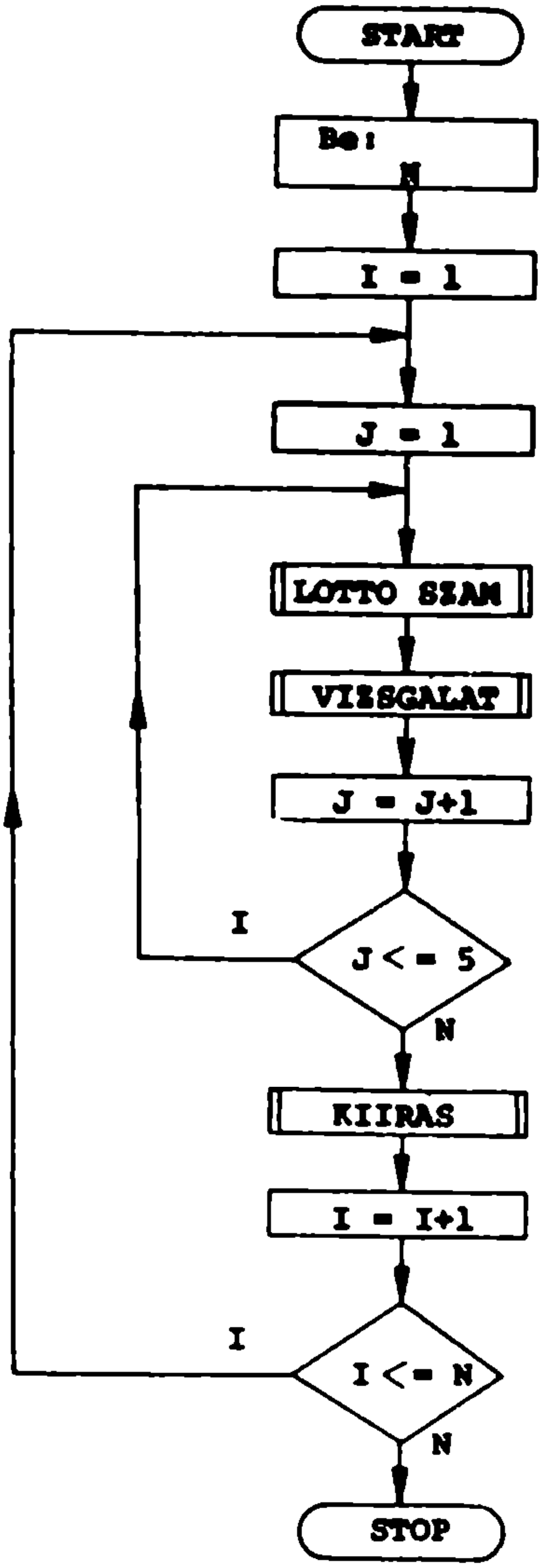
– nem kell többször leírni a programban ugyanazt az utasítássorozatot, az ilyen programrészlet lehet egy szubrutin;

– kész szubrutinokat, amik valamilyen részfeladatok megoldására szolgálnak, beépíthetünk a programunkba.

Mintaprogramunk esetén először a főprogramot írtuk meg, azután a szubrutinokat. Ezt felülről történő tervezésnek nevezzük.

● Írjunk programot, ami N darab LOTTO szelvény kitöltéséhez ad tippet!

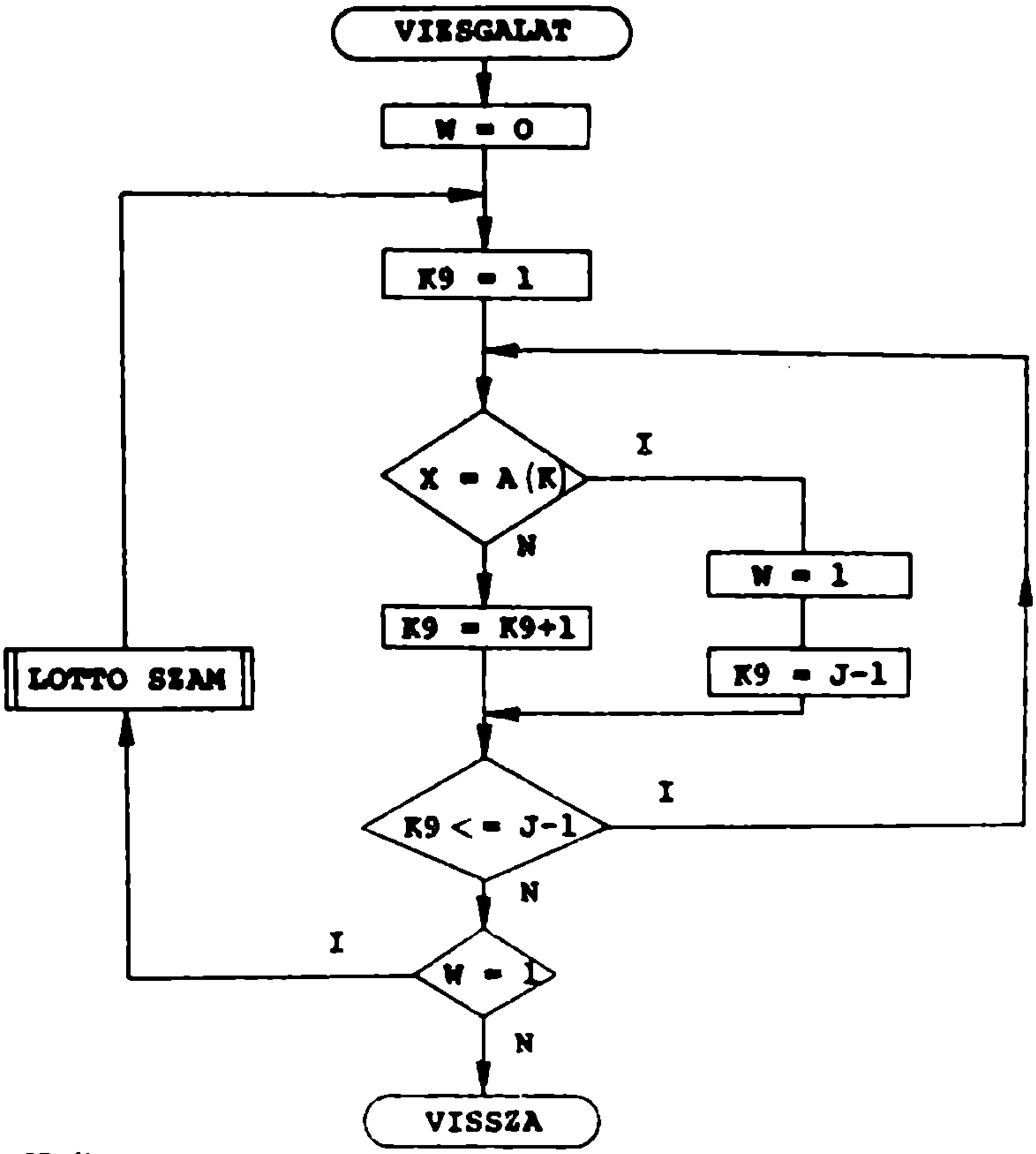
A program tervezését most is „felülről" kezdjük. Először a főprogramot gondoljuk át. A LOTTO számok előállítását, a számok egyezőségének vizsgálatát, és a sorba rendezett kiíratást szubrutinokkal oldjuk meg (56. ábra).



56. ábra

Mielőtt megírnánk a főprogramot, megtervezzük a szubrutinokat, és így alulról közelítjük meg a megoldást.

A lottószámokat az öt elemű A vektorban helyezzük el. miután megállapítottuk, hogy ilyen szám nincs a korábban előállított számok között. A W változó értékével jelezzük, ha találtunk ugyanilyen számot (57. ábra).



57. ábra

```
 10  REM LOTTO SZELVENYEK
 20  CLS:RANDOM
 30  INPUT "SZELVENYEK SZAMA",N
 40  FOR I=1 TO N
 50  FOR J=1 TO 5
 60  GOSUB 180
 70  IF J<>1 THEN GOSUB 120
 80  A(J)=X
 90  NEXT J
100  GOSUB 210:NEXT I
110  END
120  REM VIZSGALAT
130  W=0:FOR K9=1 TO J-1
140  IF X=A(K9)THEN W=1:K9=J-1
150  NEXT K9
160  IF W=1 THEN GOSUB 180:GOTO 130
170  RETURN
```

```
180  REM LOTTOSZAM
190  X=RND(90)
200  RETURN
210  REM KIIRAS
220  V=0:FOR L=1 TO 4
230  IF A(L)>A(L+1) THEN T=A(L):A(L)=
     A(L+1):A(L+1)=T:V=1
240  NEXT L
250  IF V=1 THEN 220
260  FOR L=1 TO 5:PRINT A(L),:NEXT L
270  PRINT
280  RETURN
```

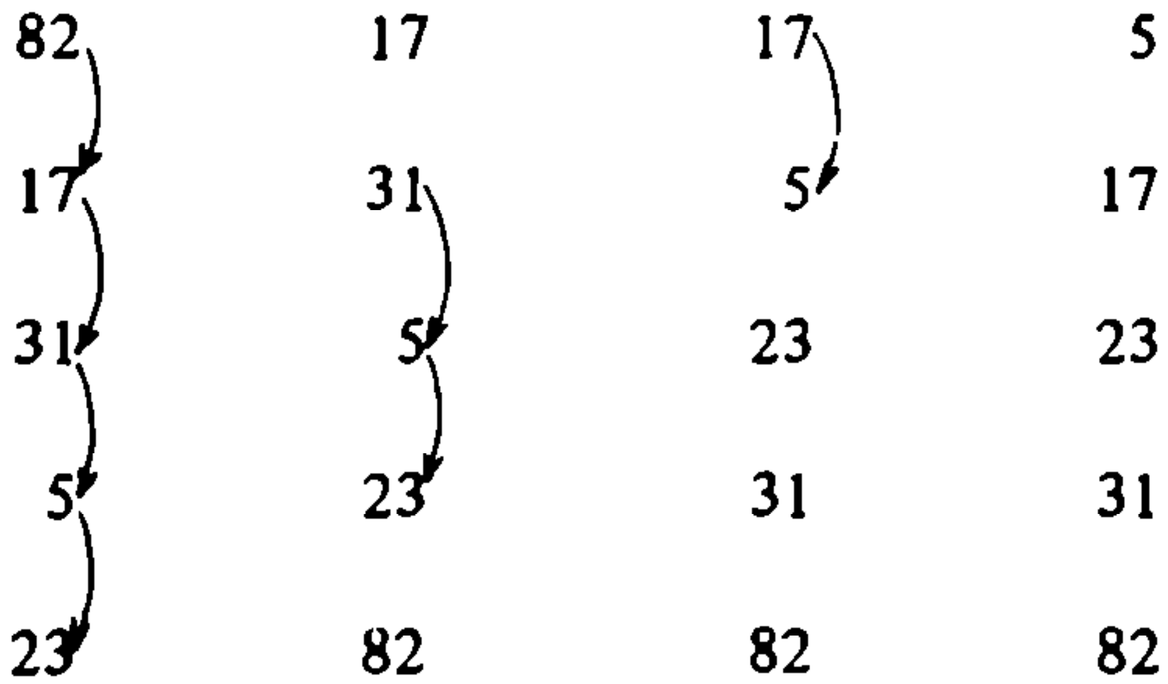
Azonos szám előfordulása esetén előállítunk egy másik lottószámot, és a vizsgálatot kezdjük elölről.

A VIZSGALAT szubrutin kezdő sorszáma: 120

– bemenő paraméterei:
  A vektor – a lottószámok értékei,
  J – az előállított lottószámok száma;
– használt változók
  K9 – ciklusváltozó,
  W – azonos szám előfordulásának jelzése;
– kimenő paramétere:
  X – új lottószám, amelyik a korábbiak közül egyikkel sem azonos.

A KIIRAS szubrutinban a lottószámokat, azaz az A vektor elemeit növekvő sorrendbe állítjuk és kiírjuk.

A sorbarendezést az ún. buborék módszerrel végezzük. A módszer lényege az, hogy a vektor szomszédos elemeit összehasonlítjuk, és értéküket kicseréljük, ha nem megfelelő a sorrend. Kövessünk végig egy ilyen rendezést! Azt látjuk, hogy a nagyobb lottószámok a magasabb indexű változók felé „mozognak", úgy mint a buborékok.



A V értéke jelzi, hogy az összehasonlítások során mikor nem történt csere. Akkor, ha nincs csere, a sorbarendezés kész.

Vegyük észre, hogy a vizsgált sorozat végén legalább egy szám biztosan a megfelelő helyre kerül. A vizsgálatot erre már nem végezzük.

A „buborék" rendezés folyamatábráját megtaláljuk az Indexes változók c. fejezet feladatai között. A szubrutin a 210-es sorszámmal kezdődik.

A KIÍRAS szubrutin

– bemenő paraméterei:

A vektor – lottószámok;

– kimenő paramétere nincs;

– használt változók:

V – csere jelzése,

L – ciklusváltozó,

T – átmeneti tároló.

Ezek után írjuk meg a lottószámokat előállító programot!

● Írjunk programot, ami N darab totószelvény kitöltéséhez ad tippeket! Az 1, X és 2 tippek előfordulásának aránya legyen 1: 2: 3!

A tippek kiírása szubrutinokkal történik. A három lehetséges eset közül a kiválasztást az ON–GOSUB utasítással oldjuk meg.

```
 10  REM TOTO TIPPEK-1
 20  CLS:INPUT"SZELVENYEK SZAMA",N
 30  RANDOM
 40  FOR I=1 TO N:FOR J=1 TO 13
 50  T=RND(6)
 60  ON T GOSUB 100,100,100,110,110,120
 70  NEXT J:PRINT:NEXT I
 80  END
100  PRINT "1":RETURN
110  PRINT "X":RETURN
120  PRINT "2":RETURN
```

Az ON–GOSUB feltételes szubrutinhívó utasítás *hatása* a GOSUB utasítással azonos, ugyanakkor *a végrehajtandó szubrutin* kiválasztása úgy történik, mint az ON–GOTO utasítás esetén.

Programunkban a T változó értéke határozza meg, hogy a GOSUB utasítás sorszámlistájának hányadik elemét választjuk. Ez lesz a végrehajtandó szubrutin kezdő sorszáma.

Amennyiben T értéke 1, 2 vagy 3 a tipp „1"-es, ha 4 vagy 5, akkor „X", ha 6, akkor „2"-es. A tippek várható előfordulásának aránya tehát 3: 2: 1 lesz.

A *feltétel* tetszőleges aritmetikai kifejezéssel megadható. Annak egész értéke határozza meg, hogy a sorszámlista hányadik eleme lesz a végrehajtandó szubrutin kezdő sorszáma. Amennyiben a kifejezés egész értéke 0 vagy nagyobb, mint a sorszámlista elemeinek száma, az ON–GOSUB utasítást követő utasítással folytatódik a program.

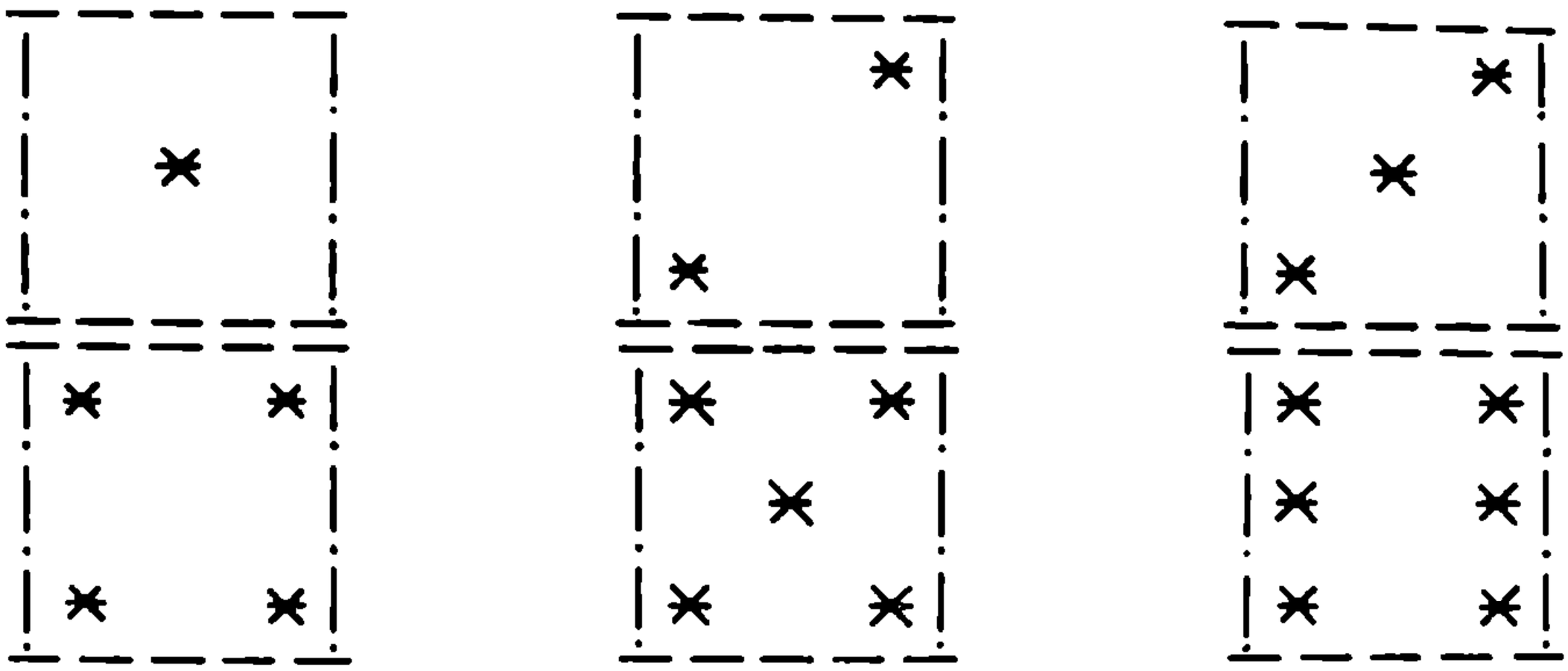
Az előzőeket figyelembe véve a következő programmal is megoldható a feladat:

```
 10  REM TOTO TIPPEK-2
 20  CLS:INPUT "SZELVENYEK SZAMA",N
 30  RANDOM
 40  FOR I=1 TO N:FOR J=1 TO 13
 50  ON 6*RND(Ø)GOSUB 100,100,100,120
 60  PRINT "X"
 70  NEXT J:PRINT:NEXT I
 80  END
```

```
100  PRINT "1";RETURN
120  PRINT "2";RETURN
```

● **Készítsünk „kockadobó" programot. A kocka oldallapjának kirajzolásával jelezzük a dobás értékét!**

Tervezzük meg az oldalakat (58. ábra).



58. ábra

Az első ötlet az, hogy írunk hat szubrutint, a hat különböző oldal kirajzolására, és a „dobás" értékének megfelelő szubrutint az ON–GOSUB utasítással választjuk ki.

Vegyük észre, hogy az oldalak rajzában azonos karakterláncok ismétlődnek. Az összes esetet figyelembe véve az oldalak kirajzolásához elegendő a következő hat rövid szubrutin:

```
200  PRINT "-----";RETURN
210  PRINT "!*   !";RETURN
220  PRINT "!   *!";RETURN
230  PRINT "!* *!";RETURN
240  PRINT "! * !";RETURN
250  PRINT "!    !";RETURN
```

A feladatra két megoldást adunk.

Azt szeretnénk ezzel bemutatni, hogy a „mintamegoldás mindig csak egy a lehetséges megoldások közül".

```
10  REM DOBOKOCKA-1
20  RANDOM:X=RND(6)
30  GOSUB 200
40  ON X GOSUB 70,80,90,100,110,120
50  GOSUB 200
60  END
70  GOSUB 250:GOSUB 240:GOSUB 250:RETURN
80  GOSUB 220:GOSUB 250:GOSUB 210:RETURN
```

```
 90  GOSUB 220:GOSUB 240:GOSUB 210:RETURN
100  GOSUB 230:GOSUB 250:GOSUB 230:RETURN
110  GOSUB 230:GOSUB 240:GOSUB 230:RETURN
120  GOSUB 230:GOSUB 230:GOSUB 230:RETURN
```

A masodik megoldas:

```
 10  REM DOBOKOCKA-2
 20  RANDOM:X=RND(6)
 30  GOSUB 200
 40  ON X GOSUB 250,220,220,230,230,230
 50  ON X GOSUB 240,250,240,250,240,230
 60  ON X GOSUB 250,210,210,230,230,230
 70  GOSUB 200
 80  END
```

Kérdések és feladatok

1. Mit nevezünk szubrutinnak?
2. Mikor ajánlott a szubrutinok használata?
3. Mit kell tudni egy kész szubrutinról, ha azt használni akarjuk a programban?
4. Értelmezzük a KIIRATAS szubrutint! Tedd alkalmassá kétszeres pontosságú értékek kiírására!
5. Írjuk át a FUGGVENYTABLAZAT programot úgy, hogy a FEJLEC szubrutin végrehajtására csak egyszer kerüljön sor!
   a) I kezdõ értéke legyen 10-nél nagyobb;
   b) a képernyõ törlését vezérlõ karakter kiírással oldjuk meg (9. melléklet)!
6. Írjunk fõprogramot, amivel a sorbarendezõ szubrutint kipróbálhatjuk!
7. Általánosítsuk a sorbarendezõ szubrutint!
8. Miért kellett END utasítást írnunk a fõprogramok végére a szubrutinok alkalmazása esetén?
9. Módosítsuk a TOTO TIPPEK programban a tippek valószínûségének arányát! Legyen az arány: 2: 1: 1!
10. Írjuk meg a TOTO TIPPEK és a DOBOKOCKA programokat indexes változók alkalmazásával!
11. Írjunk szubrutint, ami meghatározza, hogy egy karakterlánc elõfordul-e egy adott karakterláncban!
    bemenõ paraméterek:
    O$ – az a karaktersorozat, aminek az elõfordulását vizsgáljuk;
    W$ – az, amiben vizsgáljuk;
    Kimenõ paraméter:
    L9 – hányadik karaktertõl található W$-ben O$.
    L9 értéke 0, ha nem található.
12. Készítsünk programot, ami a lottó-, ill. totószelvények kiértékelését végzi! A találatok számlálásához írjunk egy olyan szubrutint, amit mindkét program használni tud!

Alkalmazási feladatok

EGY ÖSSZETETTEBB FELADAT

Vállalatok, üzemek gazdasági, termelési, értékesítési, ügyviteli és műszaki tevékenysége együtt jár nagy mennyiségű adat tárolásával és feldolgozásával. Ilyen jellegű feladatok megoldására kiválóan alkalmas a számítógép. Meggyorsíthatja az ügyintézést, segítheti a vállalati folyamatok irányítását és gyors információval szolgálhat a vállalati vezetés számára.

● Használt személygépkocsik vásárlásával és eladásával foglalkozó vállalat a következő feladatokat szeretné megoldani számítógéppel:

– a forgalom időbeni alakulásáról, személygépkocsi típusonként és összesítve készítsen kimutatást;

– adott személygépkocsi típus vásárlóinak címlistáját állítsa össze;

– adott értékhatárok közé eső személygépkocsik rendszámát írja ki;

– egy adott időpontnál régebben vásárolt gépkocsikról, a vásárlások időpontjának sorrendjében adjon listát!

Mintafeladatunkkal, – az alkalmazási lehetőség bemutatásán kívül – az összetettebb programok, programrendszerek készítését megelőző, ahhoz szorosan kapcsolódó feladatokra is szeretnénk felhívni a figyelmet. Az adatok kódolása, rekordok, adatállományok kialakítása éppúgy meghatározója a készítendő programnak, mint az alkalmazott algoritmus. Mindezek megtervezéséhez, a megfelelő módszerek kiválasztásához viszont az szükséges, hogy ismerjük azt a programozási nyelvet és számítógépet, amin meg akarjuk írni a programot és amin azt futtatni szeretnénk. Ismernünk kell lehetőségeinket.

A feladat megoldása során a következő kérdésekre kell választ adnunk:

Milyen kimenő adatokat szeretnénk kapni?

Ehhez milyen bemenő adatok szükségesek?

Milyen számítógéppel kell megoldani a feladatot?

Hogyan kódoljuk az adatokat?

Hogyan alakítsuk ki a rekordokat, adatállományokat?

Hol tároljuk az adatokat?

Milyen műveleteket kell végezni az adatokon, rekordokon, adatállományokon?

Milyen algoritmust alkalmazzunk?

Az első kérdésre a feladat megfogalmazásával adtuk meg a választ. Ez egyben meghatározza azt is, hogy milyen bemenő adatokra van szükség.

A hardver feltételek ilyen feladatoknál döntőek lehetnek. Megfelelő nagyságú memória és háttértároló szükséges az adatok tárolásához. A HT–1080Z számítógép mágneslemezt és mágnesszalagot is használhat adattárolásra. Az iskolák csak az utóbbival rendelkeznek, ezért a mintafeladatban is csak ezt használjuk.

Az adatok mágnesszalagra történő kiírása és onnét azok beolvasása nagyon lassú. (A gyakorlatban csak mágneslemezzel érdemes megoldani ezt a feladatot.)

Ezért az adatok elhelyezésével, a file szerkezetek kialakításával próbálunk javítani a helyzeten.

Mágnesszalagon adatállomány csak *soros szervezésű* lehet. Ez azt jelenti, hogy az adatállomány i. rekordját csak úgy tudjuk beolvasni, ha előzőleg beolvastuk az előző i–1 rekordot.

Célszerűnek látszik, hogy az adatállomány a gépkocsik rendszáma szerint *rendezett* legyen, azaz a rekordok a rendszámok növekvő sorrendjében helyezkedjenek el a szalagon.

A feladatok megoldásához szükséges adatok a következő elrendezésben alkotnak egy rekordot:

Adat	Karakterszám
vétel	
rendszám	6
típus kódszáma	2
vétel időpontja	
évszám utolsó két jegye	2
hónap	2
nap	2
vételi ár	6
eladási ár	6
Előző tulajdonos	
neve	változó + $
címe	
irányítószám	4
helység, utca, házszám	változó + &
eladás	
eladás időpontja	
évszám utolsó két jegye	2
hónap	2
nap	2
új tulajdonos neve	
neve	változó + *
címe	
irányítószám	4
helység, utca, házszám	változó

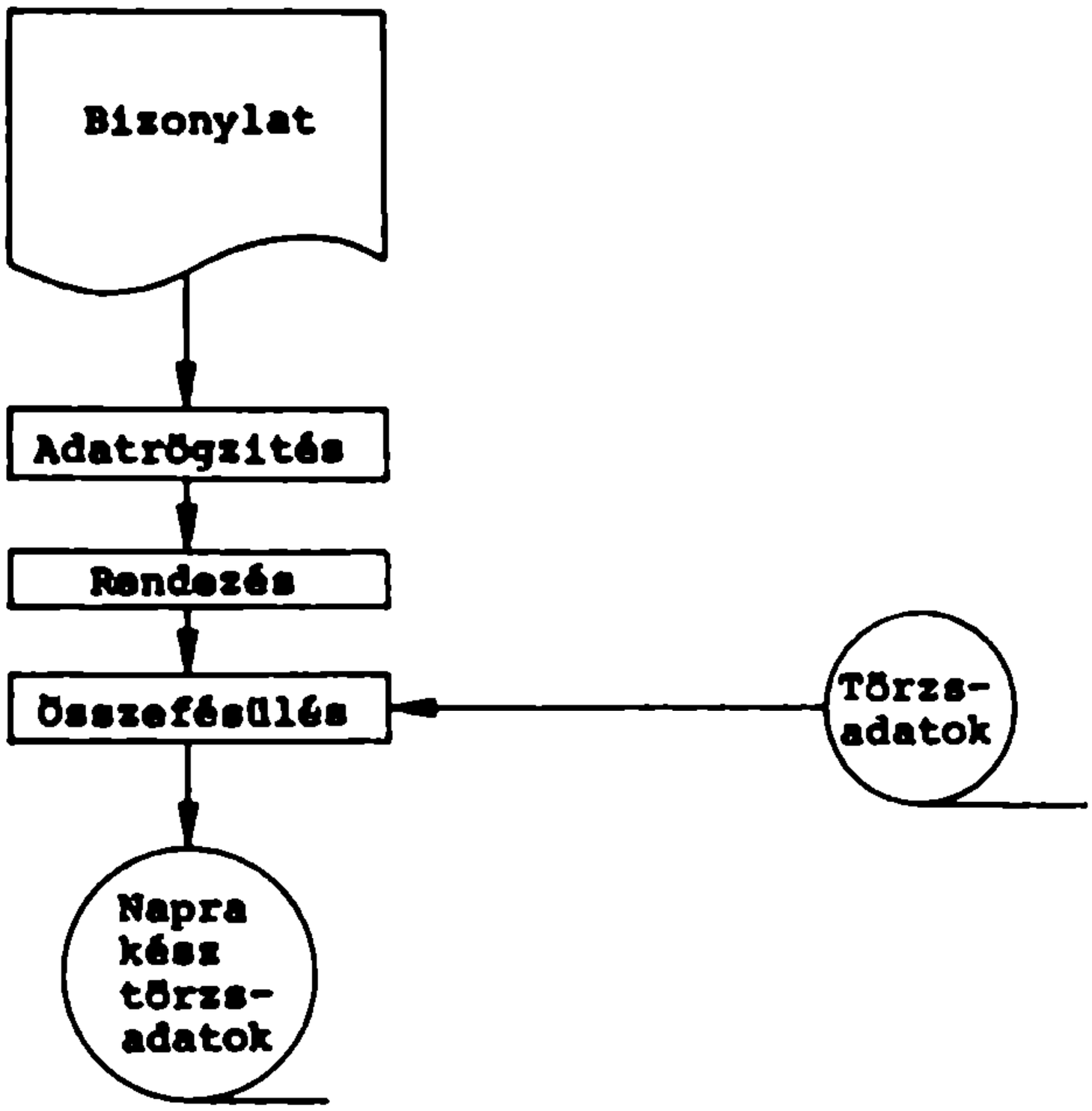
Az adatokat egy karakterlánccá összefűzzük, és úgy írjuk ki a szalagra. Feldolgozáskor fordított műveletet kell végeznünk: a karakterláncot fel kell bontani adatokra.

Amennyiben az adatok hosszát ismerjük, könnyű meghatározni, hogy hol helyezkedik el az, amelyiket keressük. Helytakarékosság szempontjából azonban nem érdemes minden adat hosszát rögzíteni. A változó hosszúságú adatok kezdetét vagy a végét, vagy mindkettőt valamilyen (máshol elő nem forduló) karakterrel jelöljük ($, &, * stb.). Pozíciójukat egy szubrutinnal határozzuk meg.

A rendszer alapja az a program, amivel a napi forgalom *adatait rögzítjük* és napra kész állapotba hozzuk a törzsadatokat tartalmazó adatállományt.

A napi adatokat, azok rendszám szerinti rendezése után, „összefésüljük" a törzsadatokkal. Ezzel az eljárással két rendezett adathalmazból egy rendezett halmazt állítunk elő.

Az adatrögzítés és a törzsadatok felújításának *rendszer folyamatábrája* a végrehajtandó tevékenységeket, azok sorrendjét, és az adatok mozgását írja le (59. ábra).



59. ábra

- Írjuk meg az adatrögzítõ programrészletet!

Az adatokat az adás-vételi szerzõdés alapján gépeljük be. Ez lesz a *bizonylatunk*. A programot úgy írjuk meg, hogy az adatok hosszára az adatrögzítõnek nem kell ügyelnie.

```
 10  REM * * * ADATROGZITES * * *
 20  CLS:CLEAR 5000:I=1
 30  PRINT "VETEL-ELADAS-ZARAS V-E-Z";
 40  V$=INKEY$:IF V$="E" THEN 170
 50  IF V$="Z" THEN 240 ELSE IF V$<>"V" THEN 30
 60  INPUT "RENDSZAM";R$
 70  PRINT "VASARLAS IDOPONTJA"
 80  GOSUB 1000:D$=X$
 90  INPUT "VETELI AR";A:Y=A
100  GOSUB 2000:A$=Y$
110  INPUT "ELADASI AR";E:Y=E
120  GOSUB 2000:E$=Y$
130  GOSUB 3000:GOSUB 4000
140  IF V$="I"THEN 60
150  A$(I)="V"+R$+T$+D$+A$+E$+N$+"$"+C$+V$+"&"
160  I=I+1 :GOTO 30
170  INPUT "RENDSZAM";R$
```

```
 180  PRINT "ELADAS IDOPONTJA"
 190  GOSUB 1000:D$=X$
 200  PRINT "UJ TULAJDONOS":GOSUB 3000
 210  GOSUB 4000:IF V$="I" THEN 60
 220  A$(I)="E"+R$+D$+N$+"*"+C$+U$
 230  I=I+1:GOTO 30
1000  REM *** DATUM ***
1010  INPUT "EV 19",P:IF A>99 THEN 1010
1020  INPUT "HO",Q:IF Q>12 THEN 1020
1030  INPUT "NAP",R:IF R>31 THEN 1030
1040  X=P*1E4+Q*1E2+R:X$=STR$(X)
1050  X$=RIGHT$(X$,6)
1060  RETURN
2000  REM *** KIEGESZITES ***
2010  Y$=STR$(1E6+Y):Y$=RIGHT$(Y$,6)
2020  RETURN
3000  REM *** NEV-LAKHELY ***
3010  INPUT "TULAJDONOS NEVE", N$
3020  INPUT "CIM-IRANYITOSZAM", C$
3030  INPUT "HELYSEG, UTCA, HAZSZAM", U$
3040  RETURN
4000  REM *** JAVITAS ***
4010  PRINT "AKAR JAVITANI I-N",
4020  V$=INKEY$:IF V$<>"I" OR
                   V$<>"N" THEN 4020
4030  RETURN
```

Numerikus adatok karakterlánccá való átalakításakor gondolnunk kell az előjel leválasztására (1050-es és 2010-es sorok). A numerikus adatokat numerikus változók értékeként kérjük, hogy csak numerikus adatot fogadjon el a program. Ezt vizsgálja a rendszerünk.

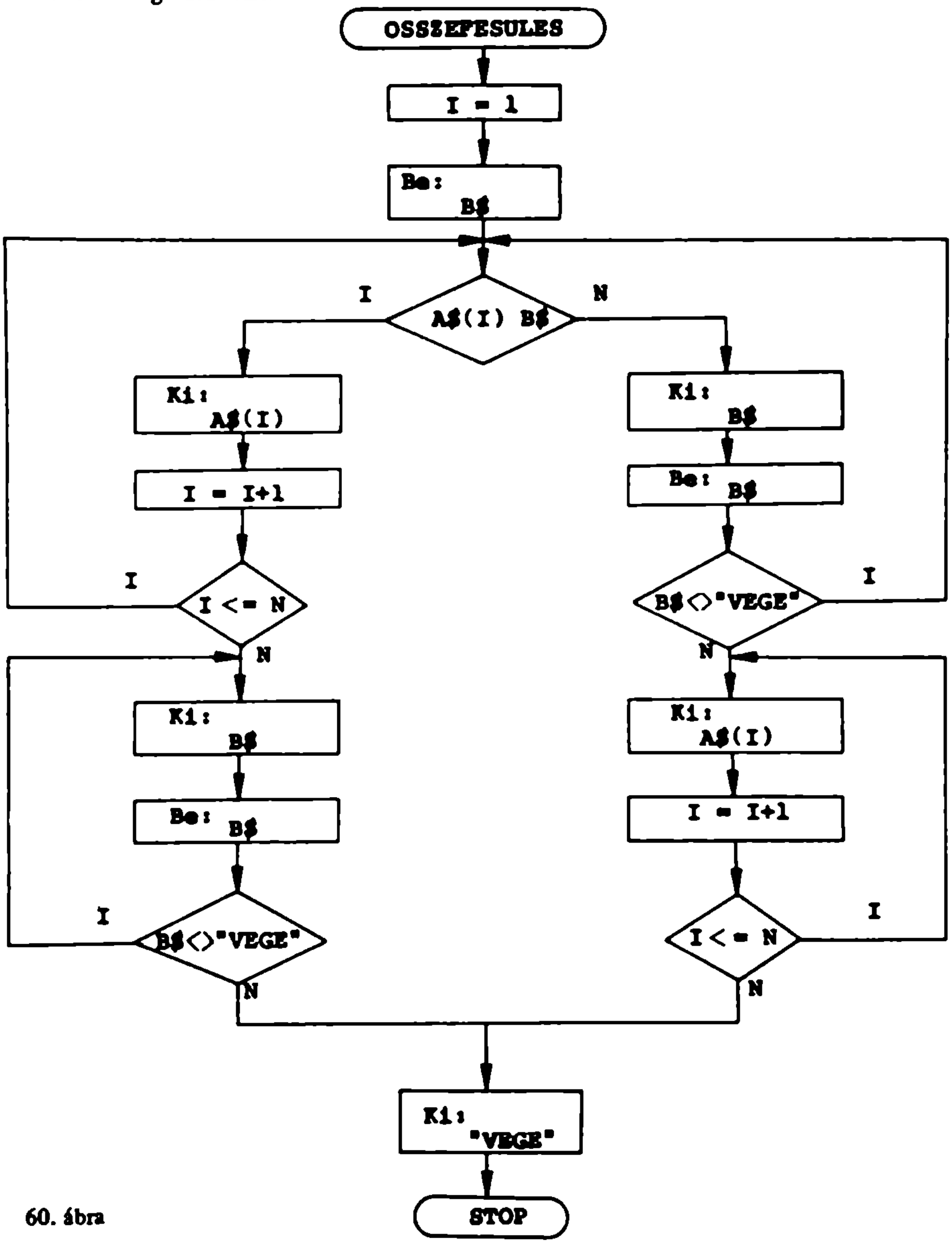
● Egészítsük ki a programot! Rendezzük az adatokat a gépkocsik rendszáma szerint növekvő sorrendbe!

```
 240  REM *** RENDEZES ***
 250  N=I-1
 260  FOR I=1 TO N-1
 270  M=I:FOR J=I+1 TO N
 280  IF MID$(A$(J),2,6) < MID$(A$(M),2,6)THEN
      M=J
 290  NEXT
 300  W$=A$(I):A$(I)=A$(M):A$(M)=W$
 310  NEXT
```

● Egészítsük ki a programot! Az A$ vektor elemekből és a mágnesszalagon tárolt törzsadatokból hozzuk létre a napra kész rendezett adatállományt !

Az „összefésülés" módszerét akkor alkalmazhatjuk, ha az adathalmazok rendezettek. Ezt a feltételt elégítettük ki az adatok rendezésével.

A törzsadatokat az 1-es számú magnetofonról olvassuk be. Mindig csak egyet, és azt összehasonlítjuk az A$ vektor soron következő elemével. Azt a rekordot íratjuk ki, amelyben a rendszám kisebb. A következő rekordot onnan választjuk ezután, ahonnan a kiírt rekordot vettük. Ezt folytatjuk mindaddig, ameddig valamelyik adathalmaz végére nem érünk. Ezek után már csak azokat a rekordokat kell kiíratnunk, amelyek a másik adathalmazból megmaradtak.



60. ábra

Az „összefésülés" algoritmusát írja le a 60. ábra. A folyamatábrában nem jelöltük a két rendszám egyenlőségének vizsgálatát.

Ezt az esetet is meg kell vizsgálni, mert a gépkocsi eladásakor ez áll fenn. Mindkét adathalmazból be kell olvasni egy-egy rekordot és a két rekordot összefűzve kell kiíratnunk.

Az A$ elemének első karaktere jelzi, hogy „eladás" vagy a „vétel" rekordról van szó. Amennyiben „vétel" rekord rendszámával van egyezőség, akkor az utalhat valamilyen hibára (ismételt bevitel), de jelentheti a törzsadat módosítását is Mi az utóbbira használtuk fel.

Egyenlőség esetén gondoljunk arra, hogy a két adatállomány végére egyszerre is érhetünk.

```
320  REM * * *OSSZEFESULES * * *
330  I=1:INPUT#-1,B$
340  IF MID$(A$(I),2,6)<>MID$(B$,1,6)THEN 410
350  Z$=RIGHT$(A$(I),LEN(A$(I))-1)
360  IF LEFT$(A$(I),1)="E" THEN PRINT#-2,B$+Z$ ELSE PRINT#-2,Z$
370  I=I+1:INPUT#-1,B$
380  IF I>N AND B$="VEGE" THEN 500
390  IF I>N THEN 480
400  IF B$="VEGE" THEN 440
410  IF MID$(A$(I),2,6)<MID$(B$,1,6) THEN 460
420  PRINT#-2,B$:INPUT#-1,B$
430  IF B$<>"VEGE" THEN 340
440  PRINT#-2,A$(I)
450  I=I+1:IF I<=N THEN 440 ELSE 500
460  PRINT#-2,RIGHT$(A$(I),LEN(A$) -1)
470  I=I+1:IF I<=N THEN 340
480  PRINT#-2,B$:INPUT#-1,B$
490  IF B$<>"VEGE" THEN 480
500  PRINT#-2,"VEGE"
```

Miután a törzsadatok napra készen rendelkezésünkre állnak, elkészíthetjük azokat a programokat, amelyek a törzsadatok feldolgozásával bizonyos információt szolgáltatnak.

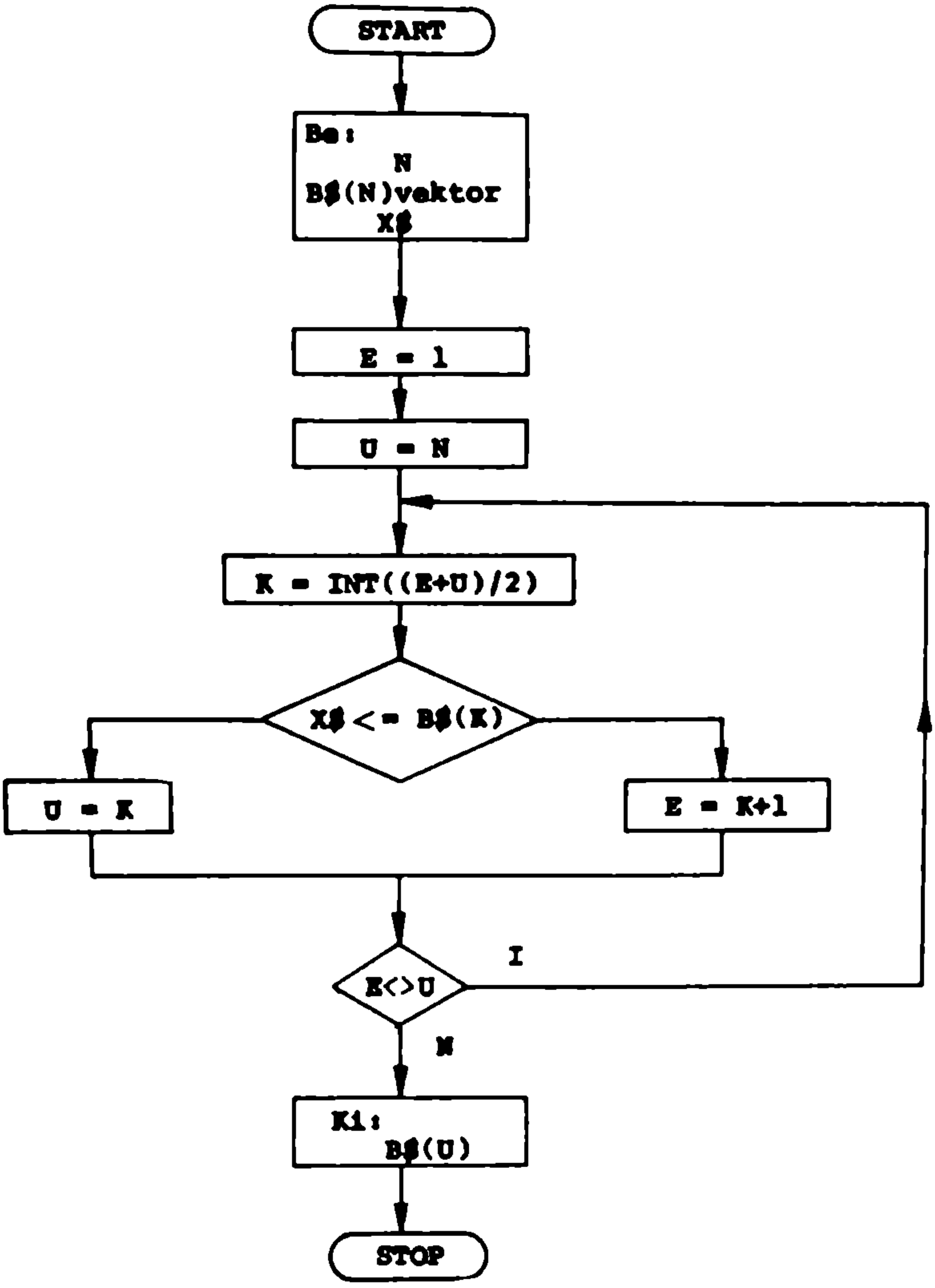
● Írjunk programot, amivel adott rendszámú gépkocsi előző és jelenlegi tulajdonosainak nevét és címét kikereshetjük!

Amennyiben nem tudunk minden adatot beolvasni a memóriába, csak az a lehetőség marad, hogy sorba minden rekordot beolvasunk és mindegyiket megvizsgáljuk. Ez a megoldás nehézkes, lassú.

Tételezzük fel, hogy a törzsadatok elférnek a memóriában. Azokat a B$ vektorban helyezzük el. Legyen a vektor elemeinek száma N, akkor a megfelelő elem kikeresése a következő algoritmus szerint történhet (61. ábra).

Ezt a módszert „felezéses keresésnek" nevezik. Csak rendezett adathalmazok esetén tudjuk használni ezt az eljárást.

Az algoritmus lényege: egy rendezett adathalmaz középső elemét kiválasztjuk



61. ábra

és összehasonlítjuk a keresett értékkel. Amennyiben a középső elem kisebb vagy azzal egyenlő, akkor a keresett értéket az adathalmaz első felében, ha nagyobb, akkor a második felében kell a továbbiakban keresni. A vizsgált adathalmaz elemeinek száma minden lépésben feleződik. Amikor a halmaznak már csak egy eleme van, a keresés véget ér. Ez lesz a keresett elem.

Ezzel az eljárással mágneslemezről is kikereshetünk adatokat.

```
10  REM * * * TULAJDONOS KIKERESESE * * *
20  CLS:CLEAR 5000
30  I=1
40  READ B$(I)
50  IF B$(I)  "VEGE" THEN I=I+1:GOTO 40
```

```
 60  INPUT "KEREM A RENDSZAMOT";X$
 70  N=I-1:E=1:U=N
 80  K=INT((E+U)/2)
 85  IF X$=B$(K)THEN U=K:GOTO 110
 90  IF X$ < B$(K) THEN U=K-1 ELSE E=K+1
100  IF U<>E THEN 80 ELSE IF X$<>B$(K) THEN PRINT
     "NEM TALALOM": END
110  M$=B$(U):K$="S":GOSUB 200
120  PRINT "ELOZO TULAJDONOS":PRINT "NEVE:",
     MID$(B$(U),26,L-26)
130  L1=L+1:K$="%":GOSUB 200
140  PRINT "CIME:",MID$(B$(U),L1,L-L1)
150  L1=L+7:K$=,"*":GOSUB 200
160  IF S=0 THEN END
170  PRINT "AZ UJ TULAJDONOS";PRINT "NEVE:",
     MID$(B(U),L1,L-L1)
180  PRINT "CIME:",RIGHT$(B$(U),LEN(B$(U))-L)
190  END
200  REM * * * KERESO SZUBRUTIN * * *
210  S=0:FOR L=1 TO LEN(M$)-LEN(K$)+1
     IF MID$(M$,L, LEN(K$))=K$ THEN S=1:RETURN
220  NEXT L:RETURN
230  DATA TP552018212221230001000000
     KOCSIS SANDOR$ 3532 MISKOLC GYAR US
     %841112 KIS LAJOS*1017 BUDAPEST
     JOKAI 3
```

Mintafeladatunk megoldását ezzel még nem fejeztük be. Írjuk meg a többi programot is, de ezt már önállóan. A számítógépet ebben a példában egy üzlet, vállalat információs rendszerébe építettük be. A bemenő adatokat az üzlet, vállalat információs rendszerétől kapjuk és a kimenő adatokat is ott használják fel. Egészítsük ki a számítógépes információs rendszerünket további programokkal, amikkel a meglevő adatokból újabb információkhoz juthatunk!

NUMERIKUS MÓDSZEREK

A számítógépek nagy műveleti sebessége olyan matematikai módszerek alkalmazását is lehetővé teszik a gyakorlatban, amelyek nagyon sok számolást igényelnek. Azokat a matematikai eljárásokat, amelyekkel véges sok lépésben, véges sok szám meghatározásával megkapjuk az eredményt, numerikus módszereknek nevezzük.

Ebben a fejezetben olyan feladatokat oldunk meg, melyek megoldására a számítási műveletek nagy száma a jellemző.

A megoldáshoz szükséges idő a számítógép műveleti sebességén kívül az algoritmus-

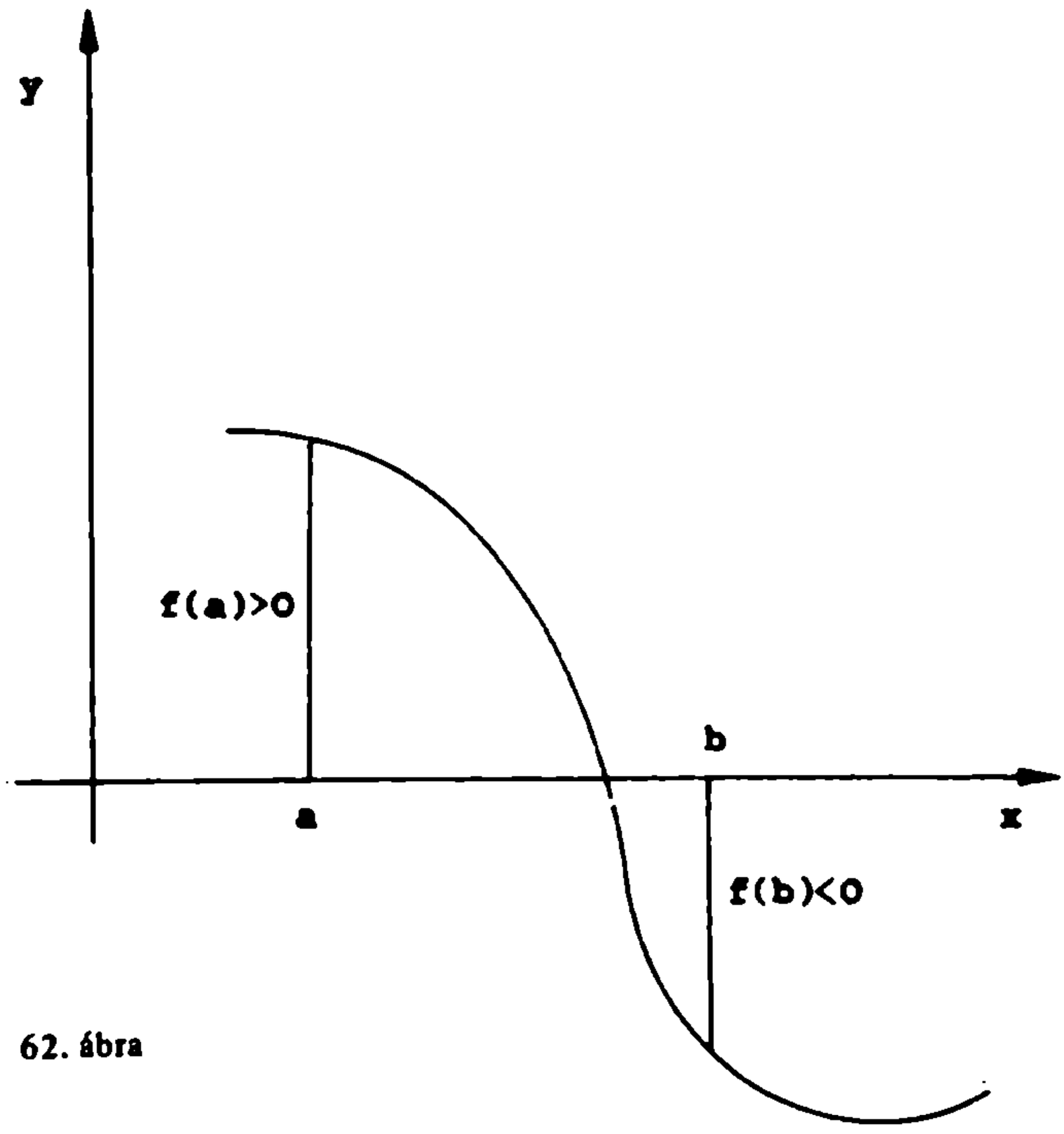
tól is függ. Egy feladat megoldására általában több algoritmust ismerünk. Ezek a gyorsaságon kívül pontosságukban is különböznek egymástól. Kiválasztásukkor ezt is figyelembe kell venni.

● Határozzuk meg az $x^4-0,8\ x^3-5,3x^2+3,8x+4,8=0$ egyenlet valós gyökeit 0,00001-nél kisebb hibával!

Ezt a feladatot középiskolai ismeretek alapján csak grafikusan tudnánk megoldani. Ezt a pontosságot azonban nem tudnánk elérni.

Az egyenlet valós gyökeinek adott pontosságú meghatározásához ismernünk kell azokat az intervallumokat, amelyekben az $y=x^4-0,8x^3-5,3x^2+3,8x+4,8$ függvény zérushelyei elhelyezkednek. Az intervallumok végpontjainak meghatározásához ábrázoljuk a függvényt, vagy alkalmazzuk a *gyökelkülönítő* programot.

A programban azt a tételt használjuk fel, ha f(a) előjele f(b) előjelével ellentétes, akkor az f(x) folytonos függvénynek biztosan van legalább egy zérushelye az (a,b) intervallumban (62. ábra). Lehet, hogy több zérushely is van, ezért a függvényt alaposan meg kell vizsgálni.



62. ábra

A gyökelkülönítést végző program csak segít a zérushelyeket tartalmazó intervallumok kikeresésében, de nem biztos, hogy minden zérushelyet meg tudunk így határozni.

A programmal a függvény értelmezési tartományának egy intervallumában, adott lépésközönként kiszámíthatjuk a függvényértékeket, és megvizsgáljuk, hogy két szomszédos x értékhez tartozó függvényérték szorzata milyen előjelű. Amennyiben a szorzat negatív, akkor a két x érték között biztosan létezik legalább egy zérushely.

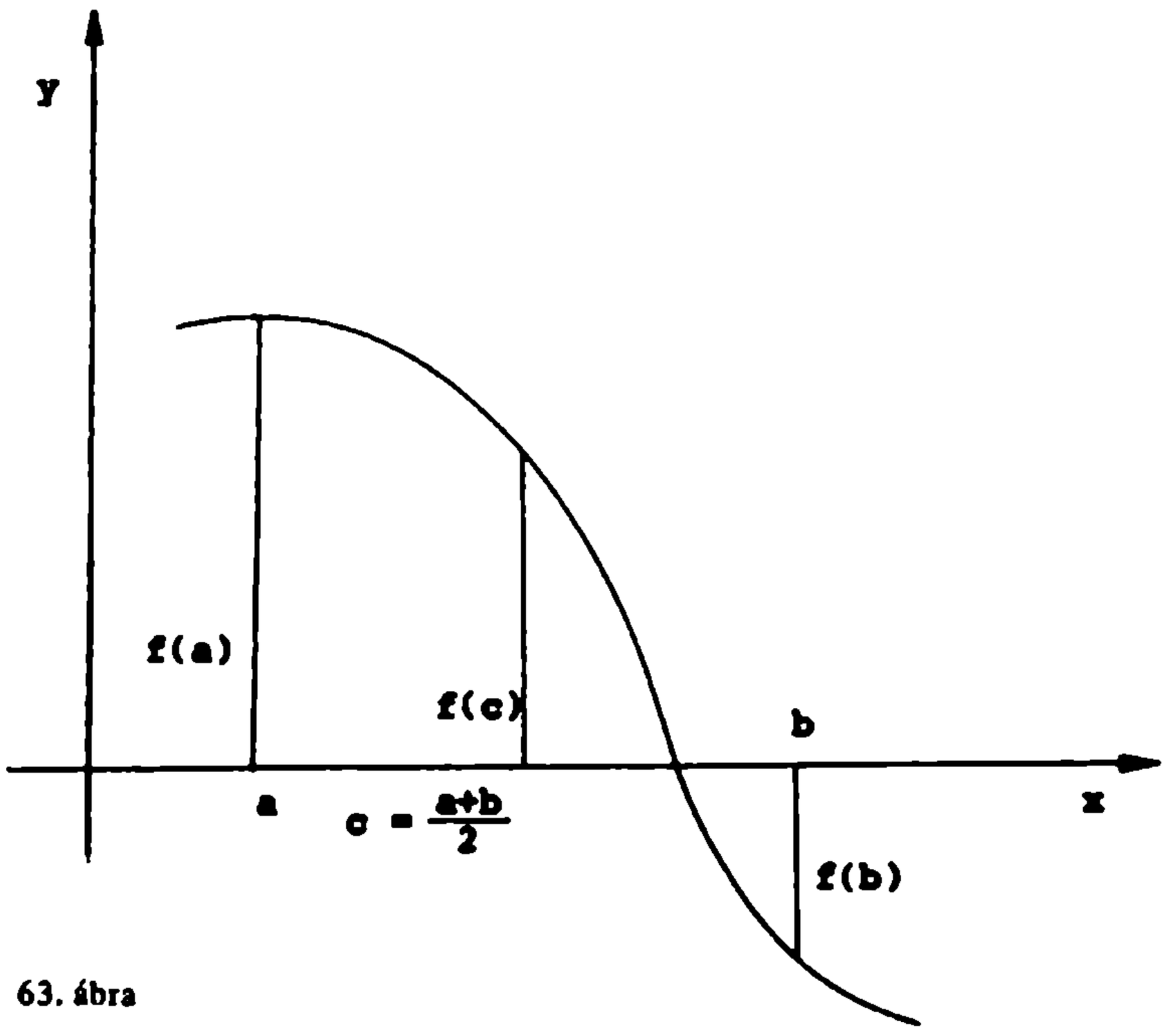
```
10  REM * * * GYOKELKULONITES * * *
20  CLS:INPUT "KEZDOPONT",K
30  INPUT "VEGPONT",V
```

```
  40  INPUT "LEPESKOZ",L
  50  X=K:GOSUB 1000:YA=Y
  60  FOR X=K+L TO V STEP L
  70  X=B:GOSUB 1000:YB=Y
  80  IF YA*YB<0 THEN PRINT
      "A=",X-L,"B=",X
  90  PRINT "YA=",YA,"YB=",YB
 100  YA=YB
 110  NEXT
 120  END
1000  REM * * *FUGGVENY * * *
1010  Y=X↑4-0.8*X↑3-5.3*X↑2+3.8*X+4.8
1020  RETURN
```

A feladatban szereplő egyenlet gyökei közül kettőt találtunk meg a programmal. Ezeket a (–3, –2) és a (–1, 0) intervallumokban kell keresnünk. Azt, hogy az egyenletnek nincs több valós gyöke, csak a függvény viselkedésének ismeretében dönthetjük el. A gyökök adott pontosságú meghatározására, a gyökfinomításra, több közelítő eljárást ismerünk. Ezek közül a legegyszerűbb az ún. *felezési módszer.*

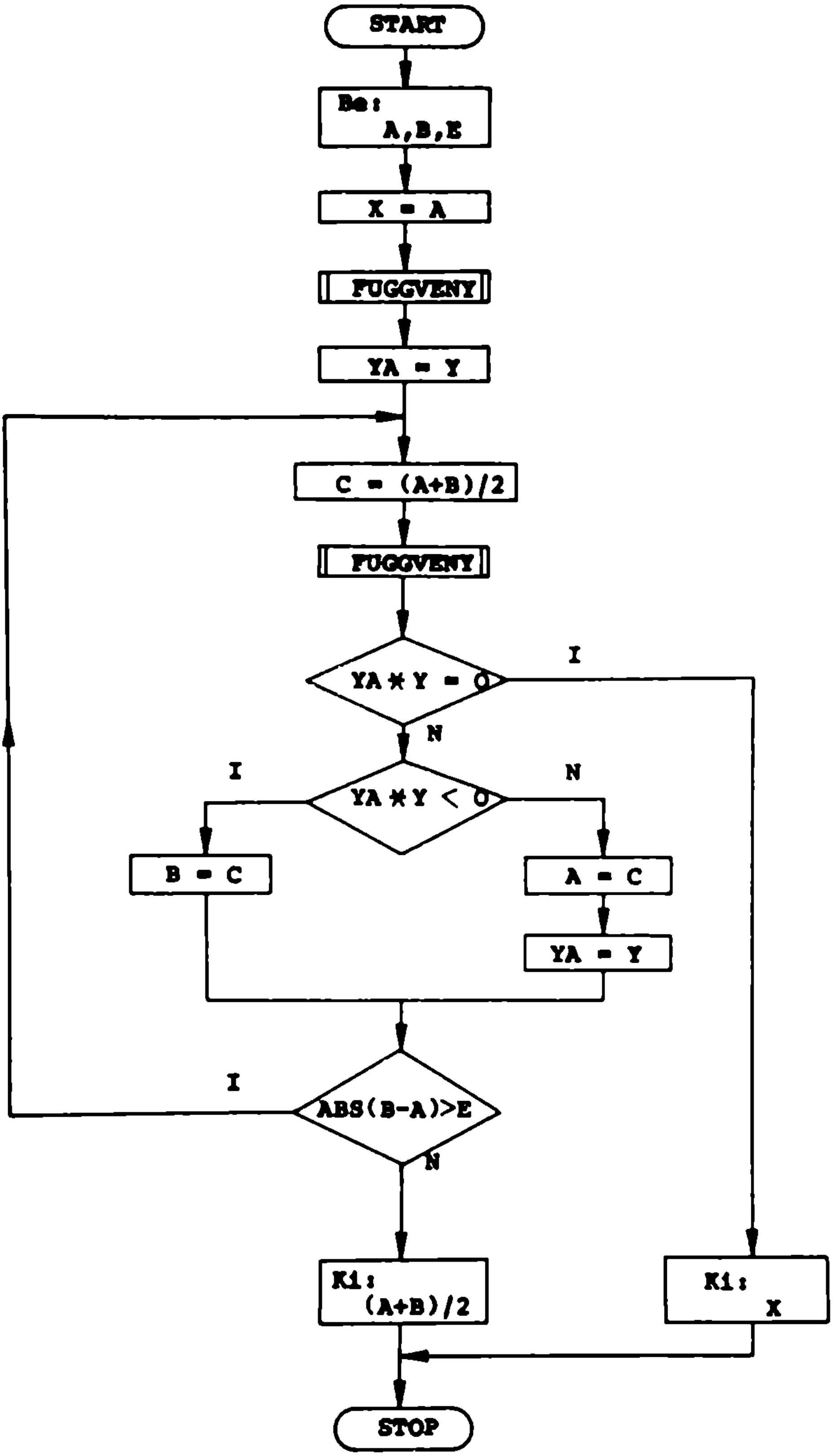
Legyen a függvénynek az (a, b) intervallumban egy zérushelyre. Határozzuk meg az intervallum c felezéspontját (63. ábra), és számítsuk ki a hozzá tartozó függvényértéket! Akkor, ha a c pont nem zérushely (ezt külön vizsgáljuk), a zérushely az (a, b) intervallum valamelyik felében van.

Mi azt vizsgáljuk, hogy f(a), f(c) különböző előjelűek-e, azaz $f(a) \cdot f(c) < 0$ egyenlőtlenség teljesül-e. Amennyiben igen, a zérushely az (a, c) intervallumban, ha nem,



63. ábra

akkor a (c, b) intervallumban van. A vizsgálat eredményétől függően c az új intervallum végpontja vagy kezdőpontja lesz. Kaptunk egy új, az előzőnél kisebb (a, b) intervallumot, amelyiknek egyik pontja a zérushely.



64. ábra

Ezt az eljárást ismételjük mindaddig, ameddig az (a, b) intervallum kisebb nem lesz egy adott értéknél, a közelítés pontosságánál.

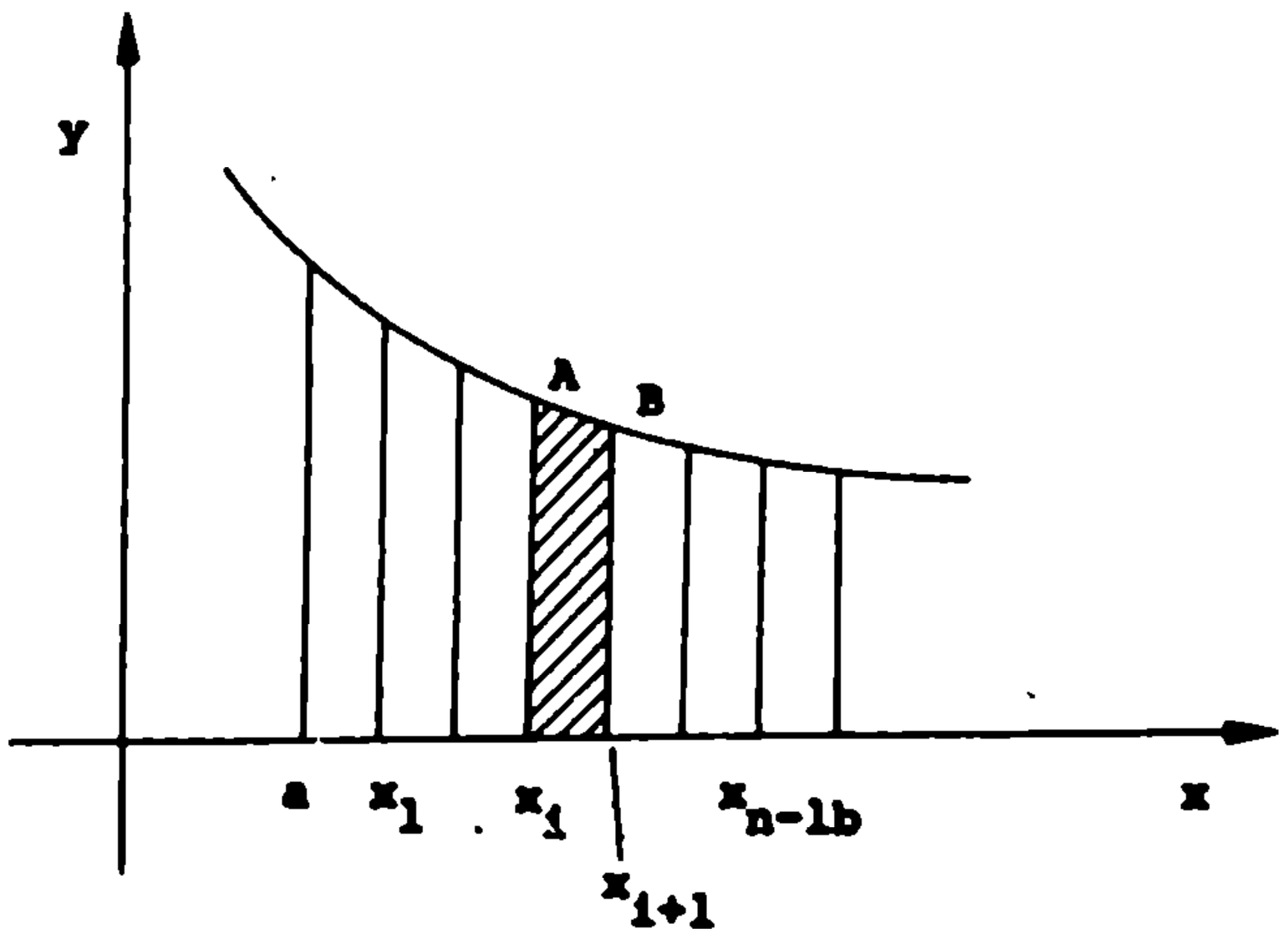
A felezési módszer folyamatábrája (64. ábra) és programja:

```
10   REM* * *GYOKFINOMITAS-FELEZES * * *
20   CLS:INPUT "AZ INTERVALLUM VEGPONTJAI·A<B",
     A,B
30   INPUT "PONTOSSAG",E
40   X=A:GOSUB 1000:YA=Y
50   C=(A+B)/2:X=C:GOSUB 1000
60   IF YA*Y=0 THEN PRINT "ZERUSHELY=",C:END
70   IF YA*Y<0 THEN B=C ELSE A=C:YA=Y
80   PRINT "A=",A,"B=",B
90   IF ABS(B-A)>E THEN 50
100  PRINT "ZERUSHELY=",FIX((A+B)/2/E)*E:END
1000 REM * * *FÜGGVENY * * *
1010 Y=X↑4-0.8*X↑3-5.3*X↑2+3.8*X+4.8
1020 RETURN
```

• Határozzuk meg az $f(x) > 0$ függvény görbéje alatti területet az [a, b] intervallum felett adott pontossággal!

Osszuk fel az [a, b] intervallumot n részre, és jelöljük az osztópontokhoz tartozó függvényértékeket. Két szomszédos osztópont x_i és x_{i+1}, valamint az $A(x_i, f(x_i))$, $B(x_{i+1}, f(x_{i+1}))$ pontok egy trapézt alkotnak (65. ábra). Az így megrajzolt trapézok területének összege közelítőleg a függvény görbe alatti területét adja. Ott követünk el pontatlanságot, hogy a függvény görbéje helyett egyenessel kötjük össze a függvény két pontját. Az eltérés egyre kisebb lesz, ha az n értékét növeljük.

Az i. részintervallum fölé emelt trapéz területe tehát:



65. ábra

$$t_i = (x_i - x_{i-1}) \cdot \frac{f(x_{i-1}) + f(x_i)}{2},$$

ahol $x_i - x_{i-1}$ a trapéz magassága.

Az [a, b] intervallumot n részre osztottuk, ezért a kis trapézok magassága $m = \frac{b-a}{n}$.

A trapézok területének összege tehát:

$$T = m\left(\frac{f(a) + f(x_1)}{2} + \frac{f(x_1) + f(x_2)}{2} + \ldots + \frac{f(x_i) + f(x_{i+1})}{2} + \ldots + \right.$$

$$\left. + \frac{f(x_{n-2}) + f(x_{n-1})}{2} + \frac{f(x_{n-1}) + f(b)}{2}\right).$$

A lehetséges összevonások végrehajtása után:

$$T = m\left(\frac{f(a)}{2} + f(x_1) + f(x_2) + \ldots + f(x_i) + \ldots + f(x_{n-2}) + f(x_{n-2}) + \frac{f(b)}{2}\right)$$

Ebből már látszik, hogyan kell megírnunk a programot. A belső osztópontokhoz tartozó függvényértékek összegét kell képeznünk T értékének kiszámításához.

A beosztások számát mindig a kétszeresére növeljük, és a két egymás után kiszámított területértéket összehasonlítjuk. Amennyiben az eltérés kisebb, mint a kívánt pontosság, akkor az utolsó összeget elfogadjuk. Ez lesz a kérdéses terület adott pontosságú mérőszáma.

```
 10  REM* * *FUGGVENY GORBEJE ALATTI TERULET * * *
 20  CLS:INPUT "INTERVALLUM KEZDOPONTJA",A
 30  INPUT "VEGPONTJA",B
 40  INPUT "PONTOSSAG",E
 50  INPUT "BEOSZTASOK SZAMA",N
 60  S=0
 70  T=0:D=(B-A)/N
 80  X=A:GOSUB 160:T=T+Y/2
 90  FOR X=A+D TO B-DSTEP D
100  GOSUB 160:T=T+Y
110  NEXT X
120  GOSUB 160:T=T+Y/2 :T=T*D
130  PRINT "N=",N,"T=",T
140  IF ABS(S-T)>E THEN S=T:N=2*N:GOTO 70
150  END
160  REM * * *FUGGVENY * * *
170  Y=1/ X
180  RETURN
```

• Írjunk programot többismeretlenes elsőfokú egyenletrendszerek megoldására!

Az algoritmust egy konkrét példán mutatjuk be. Legyen a megoldandó egyenletrendszer a következő:

$$(1) \quad 2{,}5x_1 + 3x_2 - 4x_3 = -3{,}5$$
$$(2) \quad 2x_1 + x_2 - 1{,}5x_3 = -2{,}75$$
$$(3) \quad -3x_1 - 2{,}5x_2 + 3x_3 = 3$$

Az (1) egyenletet elosztjuk 2,5-del, az x_1 ismeretlen együtthatójával. Az osztás után az első egyenlet $(1)'$ $x_1 + 1{,}2x_2 - 1{,}6x_3 = 1{,}4$ alakú lesz. A másik két egyenletet redukáljuk. Az egyenlő együtthatók ismert módszerével kiküszöböljük a x_1 ismeretlent tartalmazó tagokat a (2) és a (3) egyenletből. A (2) egyenletből az $(1)'$ egyenlet 2-szeresét, a (3) egyenletből a −3-szorosát vonjuk ki.

A redukált egyenletrendszerünk ekkor a következő:

$$(1)' \quad x_1 + 1{,}2x_2 - 1{,}6x_3 = -1{,}4$$
$$(2)' \quad -1{,}4x_2 + 1{,}7x_3 = 0{,}05$$
$$(3)' \quad 1{,}1x_2 - 1{,}8x_3 = -1{,}2$$

A $(2)'$ és a $(3)'$ egyenletben már nem szerepel az x_1 ismeretlen.

Folytatva az eljárást, a $(3)'$ egyenletből az x_2 ismeretlent küszöböljük ki:

$$(2)'' \quad x_2 - 1{,}21429x_3 = -0{,}0357143$$
$$(3)'' \quad -0{,}464286x_3 = -1{,}16071$$

Legvégül a $(3)''$ egyenletet elosztjuk az x_3 ismeretlen együtthatójával, és megkapjuk a következő egyenletrendszert:

$$x_1 + 1{,}2x_2 - 1{,}6x_3 = -1{,}4$$
$$x_2 - 1{,}21429x_3 = -0{,}0357143$$
$$x_3 = 2{,}5$$

ami az eredetivel ekvivalens.

Az utolsó egyenlet megadja az x_3 ismeretlen értékét. Ezt behelyettesítjük az előző egyenletbe, és meghatározhatjuk x_2 értékét, majd x_2 és x_3 ismeretében az első egyenletből x_1 értékét. Az egyenlet gyökei: $x_1 = -1$, $x_2 = 3$, $x_3 = 2{,}5$.

Fogalmazzuk meg általánosan, hogy melyek azok a részfeladatok, amiket meg kell oldanunk a programunkban! Ezek végrehajtására szubrutinokat írunk.

Jelöljük ki i. egyenlet j. ismeretlenjének együtthatóját a_i-vel, az i. egyenlet konstansát b_j-vel, a j. ismeretlent x_j-vel ($i = 1,2,\ldots, n$; $j = 1,2,\ldots, n$). Az i. egyenletünk ekkor a következő:

$$a_{i,1}\, x_1 + a_{i,2}\, x_2 + \ldots + a_{i,j} x_j + \ldots + a_{i,n}\, x_n = b_i.$$

OSZTAS szubrutin

Az i. egyenletet el kell osztani az i. ismeretlen együtthatójával, amennyiben az nem nulla. Vesszővel jelölve az együtthatók és a konstansok új értékeit:

$$a'_{i,j} = \frac{a_{i,j}}{a_{i,i}} : b'_i = \frac{b_i}{a_{i,i}}, \quad \text{ahol } a_{i,i} \neq 0.$$

Akkor, ha $a_{i,i} = 0$, de $a_{k,i} \neq 0$ létezik, ahol $k > i$, akkor az i. és k. oszlopot ki kell cserélnünk. Akkor, ha minden $a_{k,i} = 0$ ($k = i,\ldots, n$), az egyenletrendszernek nincs megoldása. Ennek vizsgálatával a programban nem foglalkozunk. $a_{i,i} = 0$ esetén a program leáll.

REDUKALAS szubrutin

Az i + 1. egyenlettől kezdődően minden egyenletből – jelöljük ezeket k-val – ki kell kü-

szöbölnünk az i. ismeretlent. Ezt úgy érjük el, hogy a k. egyenlet i. együtthatójával megszorozzuk az i. egyenletet, és azt kivonjuk a k. egyenletből.

$$a'_{k,j} = a_{k,j} - a_{k,i} \cdot a_{i,j};$$

$$b'_k = b_k - a_{k,i} \cdot b'_i, \text{ ahol } k = i + 1,\ldots n;$$

$j = i,\ldots, n$. Az indexelésből következik, hogy $k < n$ feltételnek teljesülnie kell, azaz $i = n$ esetben nincs redukálás.

GYOKOK szubrutin

$x_n = b_n$-nel. Az x_i-t úgy számítjuk ki, ha $x_{i+1}, \ldots\ldots, x_n$ gyököket ismerjük, hogy az i. konstansból kivonjuk az ismert gyökök és az i. egyenlet megfelelő együtthatóinak szorzatából képzett összeget.

$$x_i = b_i - (a_{i,n}x_n + a_{i,n-1}\ x_{n-1} \quad a_{i,i+1}\ x_{i+1}),$$

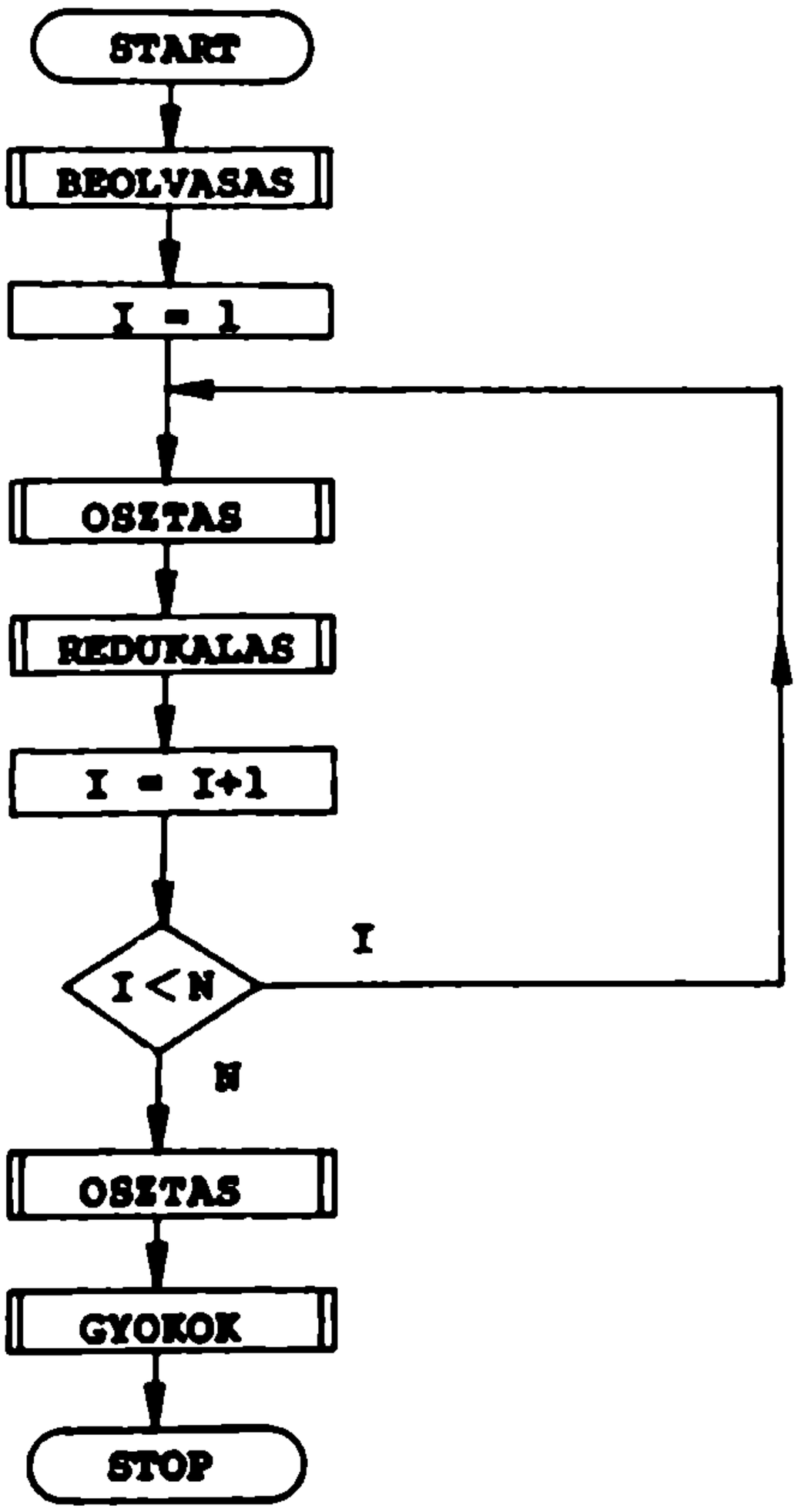
ahol $i = n-1, n-2, \ldots, 1$.

Az ismeretlenek száma N változó értéke lesz, az együtthatómátrix elemeit az A kétdimenziós tömbbe, a konstansokat a B vektorba helyezzük el.

A beolvasást a BEOLVASO szubrutin végzi. Az adatokat DATA adatlistában helyezzük el.

A KIIRO szubrutinnal az együtthatók és a konstansok aktuális értékeit íratjuk ki.

A főprogramunk folyamatábrája (66. ábra):



66. ábra

A többismeretlenes egyenletrendszer megoldására szolgáló program a következő:

```
  10  REM* * *EGYENLETRENDSZER * * *
  20  CLS:READ N:DIM A(N,N),B(N)
  30  GOSUB 1000:REM BEOLVASAS
  40  PRINT "AZ EGYENLET:":GOSUB 2000:REM KIIRATAS
  50  I=1
  60  GOSUB 3000:REM OSZTAS
  70  GOSUB 4000:REM REDUKALAS
  80  I=I+1:IF I<N THEN 60
  90  GOSUB 3000 :GOSUB 4050
 100  GOSUB 5000:REM GYOKOK
 110  END
1000  REM* * *BEOLVASAS * * *
1010  FOR I=1 TO N:FOR J=1 TO N
1020  READ A(I,J)
1030  NEXT:READ B(I):NEXT
1040  RETURN
2000   REM * * *KIIRAS * * *
2010  FOR P=1 TO N:FOR Q=1 TO N
2020  PRINT A(P,Q),TAB(9*Q),
2030  NEXT:PRINT B(P):NEXT
2040  RETURN
3000  REM * * *OSZTAS * * *
3010  R=A(I,I):IF R=0 THEN GOSUB 2000: PRINT
      "A(I,I)=0":END
3020  FOR J=I TO N
3030  A(I,J)=A(I,J)/R
3040  NEXT:B(I)=B(I)/R :NEXT
3050  RETURN
4000  REM * * *REDUKALAS * * *
4010  FOR K=I+1 TO N
4020  E=A(K,I):FOR J=I TO N
4030  A(K,J)=A(K,J)-E*A(I,J)
4040  NEXT:B(K)=B(K)-E*B(I):NEXT
4050  CLS:PRINT "A REDUKALT EGYENLET:":
      GOSUB 2000
4060  RETURN
5000  REM* * *GYOKOK * * *
5010  X(N)=B(N)
5020  FOR I=N-1 TO 1 STEP -1
5030  X(I)=B(I):FOR J=I+1 TO N
```

```
5040  X(I)=X(I)-A(I,J)*X(J)
5050  NEXT:NEXT
5060  REM * * * GYOKOK * * *
5070  CLS:PRINT"GYOKOK"
5080  FORI=1 TO N
5090  PRINT"X(",I,")=",X(I)
5095  NEXT:RETURN
6000  REM * * * ADATOK * * *
6010  DATA 3
6020  DATA 2,5,3,-4
6030  DATA 2,1,-1,5
6040  DATA -3,-2,5,3
```

Ezt az eljárást Gauss-féle kiküszöbölési módszernek nevezik.

KOMBINATORIKUS FELADATOK

Az előző fejezetben tárgyalt módszerekhez hasonlóan a kombinatorikus feladatok megoldásában is döntő az algoritmus.

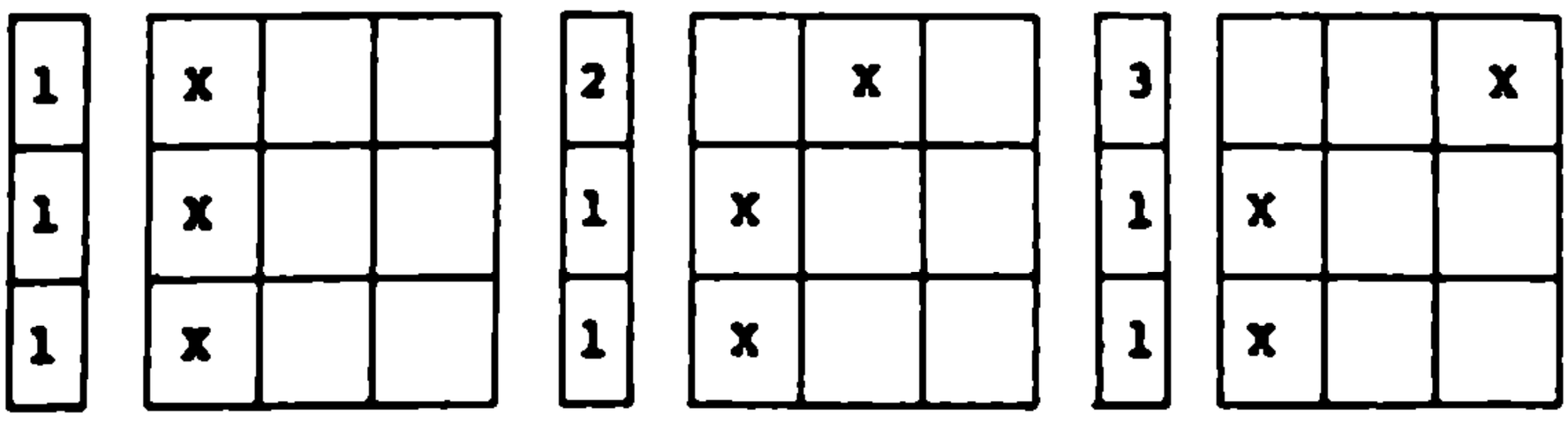
Egy optimumszámítási feladatot az összes lehetséges eset előállításával oldunk meg. Az esetek száma még kis számú adat esetén is nagy. Ilyen módszerre csak számítógép alkalmazásával gondolhatunk.

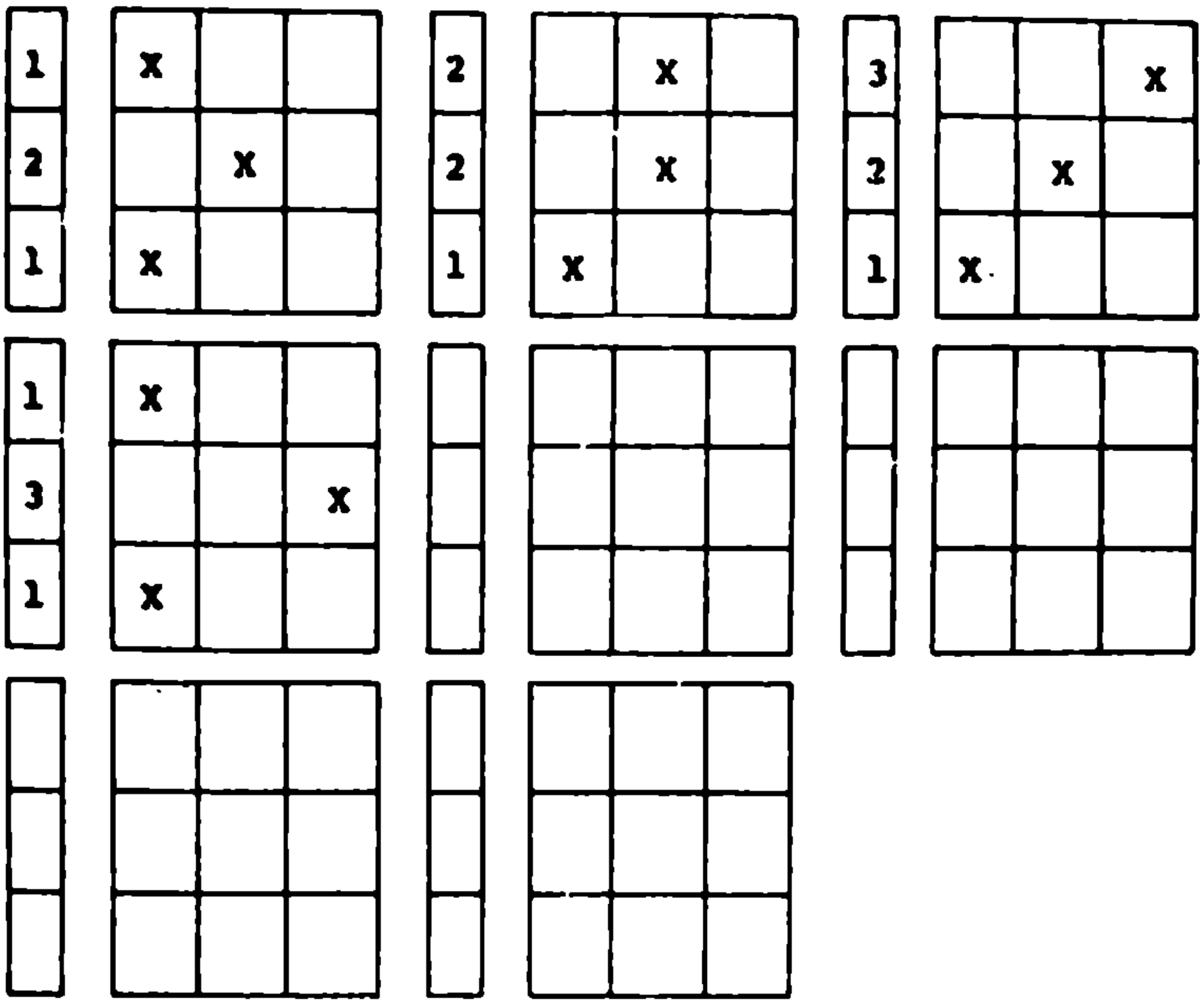
A kombinatorika témakörébe tartozik ugyan ez a feladat, de annak matematikai megfogalmazásához, a rendszer matematikai modelljének megalkotásához, a programtervezéshez gráfokat és mátrixokat is alkalmazunk.

- N várost szeretnénk a legrövidebb időn belül körbejárni. A városokat összekötő utak megtételéhez szükséges időket ismerjük. Az oda és a vissza út időtartama általában nem egyenlő. Határozzuk meg a körbejárás sorrendjét!
- Helyezzük el egy NXN-es sakktáblán N bástyát úgy, hogy minden sorban és minden oszlopban csak egy bástya legyen! Írassuk ki az összes lehetséges állást!

Két, látszólag egymástól távol álló problémáról van szó. Az első feladat gyakorlati jelentősége egyértelmű. Megoldásához a második feladat algoritmusát használjuk fel, egy-két kiegészítéssel.

Legyen a sakktáblánk 3x3-as. Az első 7 állás a következő (67. ábra):





67. ábra

Vegyük észre az előállítási szabályt, és rajzoljuk be az üres táblákba is a bástyák helyzetét!

A táblák melletti négyzetekbe azt írjuk, hogy a bástya hányadik oszlopon áll. Alulról felfelé olvasva a számokat azt látjuk, hogy mindig három egymást követő számot alkotnak a számjegyek.

Algoritmusunk tehát az, hogy a legfelső sorban elhelyezett bástyát mindig egy pozícióval jobbra toljuk. Abban az esetben, ha lekerülünk a tábláról, a bástyát az első oszlopra helyezzük, és a következő sorban lépünk eggyel jobbra. Amikor valamelyik sorban lelépünk a tábláról, mindig meg kell vizsgálnunk, hogy a következő sorban hová került a bástya. Akkor, ha a bástya a táblán van, megkaptuk a bástyák elhelyezésének egy esetét.

A táblák melletti számokkal a bástyák helyzetét egyértelműen meghatározhatjuk. Azt jelölik, hogy hányadik oszlopban állnak. Legyenek ezek az A vektor elemeinek értékei.

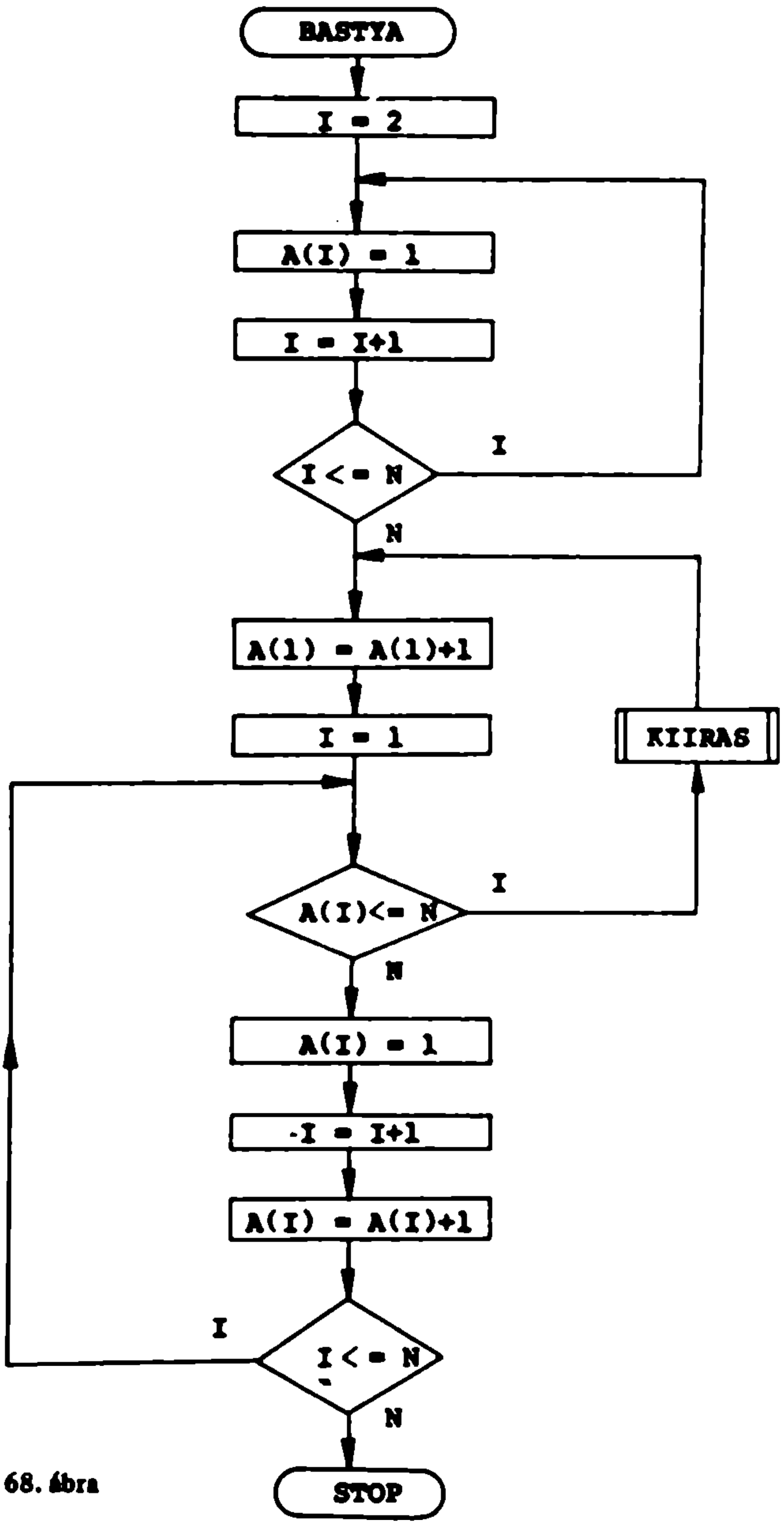
A(1) értékét kell növelnünk, és meg kell vizsgálnunk az A vektor elemeit, hogy az értékük nagyobb-e N-nél, ha a „sakktábla" NXN-es.

A kezdőértékek beállítása után a bástyák lehetséges elhelyezését követhetjük nyomon a következő folyamatábrán (68. ábra).

A bástyák elhelyezése akkor fejeződik be, ha $I > N$ reláció teljesül. Az $A(I) = A(I)+1$ utasítás végrehajtására $I = N+1$ esetében is sor kerül. Ez indokolja a DIM A(N+1) utasítás használatát.

```
10  REM* * *BASTYA * * *
20  N=8:DIM A(N+1)
30  FOR I=2 TO N:A(I)=1:NEXT
40  A(1)=A(1)+1
50  I=1
60  IF A(I)<=N THEN GOSUB 200:GOTO 40
```

```
 70  A(I)=1:I=I+1:A(I)=A(I)+1
 80  IF I<=N THEN 60
 90  END
200  REM* * * KIIRAS * * *
210  FOR L=1 TO N:PRINT A(L); :NEXT
220  PRINT
230  IF INNEY$="" THEN 230
240  RETURN
```



68. ábra

• Helyezzük el a bástyákat úgy, hogy egyik se üsse a másikat!

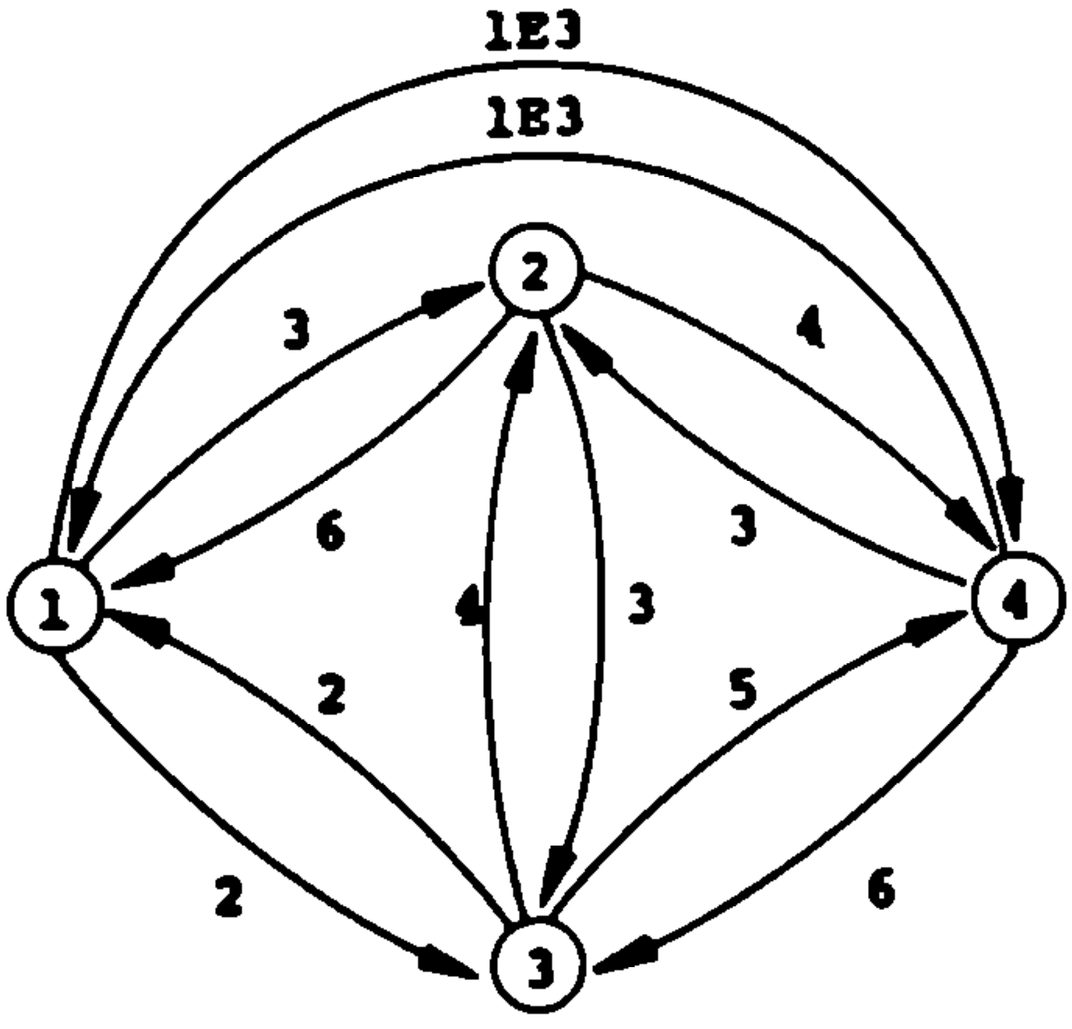
Ez azt jelenti, hogy egy oszlopban két bástya nem lehet. Ki kell szűrnünk ezeket az állásokat. Összehasonlítjuk az A vektor elemeit, és ha két elem értéke megegyezik, akkor azt az esetet nem írjuk ki.

```
200  REM * * *KIIRAS-NEM UTIK EGYMAST * * *
210  FOR J=1 TO N-1:FOR K=J+1 TO N
220  IF A(J)=A(K)THEN RETURN
230  NEXT:NEXT
240  FOR L=1 TO N:PRINT A(L),:NEXT
250  PRINT
260  RETURN
```

• Négy várost és a városokat összekötő utak megtételéhez szükséges időket adjuk meg a 69. ábrán. Nagy időértékkel jelöljük, ha a két város között nincs út.

Határozzuk meg azt az utat, ahogyan a négy várost a legrövidebb idő alatt körbejárhatjuk!

Az időértékeket helyezzük el egy mátrixba (70. ábra).



69. ábra

	1	2	3	4
1	0	3	2	1E3
2	6	0	3	4
3	2	4	0	5
4	1E3	3	6	0

70. ábra

A körutazás során minden városon egyszer haladunk át. Ezt azt jelenti, hogy egyszer megyünk be a városba, tehát minden oszlopból egyet, és egyszer jövünk ki a városból, tehát minden sorból ugyancsak egy elemet kell kiválasztanunk. Ez megfelel a bástyák olyan elrendezésének, amikor azok nem ütik egymást: A(J) = A(K).

A körutak kikeresésekor azokat az eseteket is ki kell hagynunk, amikor valamelyik elem a mátrixnak abban az átlópontjában van, amelyiknek a sor- és oszlopindexe megegyezik: A(J) = J. Ilyen elemet azért nem választhatunk, mert minden városból ki kell jönnünk.

Ugyanerre az átlóra nem lehet két kiválasztott elem szimmetrikus: A(J) = K AND A(K) = J, mert a két várost összekötő utat csak egyszer tehetjük meg.

```
 10  REM* * *KORUT * * *
 20  N=4:DIM A(N+1),B(N;N)
 30  REM* * *BEOLVASAS * * *
 40  FOR I=1 TO N:FOR J=1 TO N
 50  READ B(I,J)
 60  NEXT:NEXT
 70  REM* * *KIVALASZTAS* * *
 80  FOR I=2 TO N:A(I)=1:NEXT
 90  A(1)=A(1)+1
100  I=1
110  IF A(I)<=N THEN GOSUB 200:GOTO 90
120  A(I)=1:I=I+1:A(I)=A(I)+1
130  IF I<=N THEN 110
140  END
200  REM* * *ELLENORZES * * *
210  FOR J=1 TO N-1:FOR K=J+1 TO N
220  IF A(J)=J OR A(J)=A(K)OR A(J)=K ANDA(K)=J
     THEN RETURN
230  NEXT:NEXT
240  REM * * *OSSZEGZES * * *
250  S=0:FOR L=1 TO N:S=S+B(L,A(L)):PRINT
     A(L),:NEXT
260  PRINT S
270  RETURN
300  REM* * *ADATOK * * *
310  DATA 0,3,2,1E3,6,0,3,4
320  DATA 2,4,0,5,1E3,3,6,0
```

A feltételek teljesülését a 220-as sorban vizsgáljuk.

Az utak megtételéhez szükséges idõadatokat a DATA adatlistából olvassuk be a 4x4 méretű B tömbbe, és a körutak megtételéhez szükséges időt ebből határozzuk meg.

A program kiírja az összes körutat. A legkisebb időérték kiválasztását a program kiegészítésével oldhatjuk meg.

SZIMULÁLÁS

Az előző fejezetben egy „kirándulás" matematikai modelljét adtuk meg. A számítógép segítségével minden lehetséges útvonalat kipróbáltunk. Egy-egy úthossz kiírását tekinthetjük úgy is, mintha az útvonalat végigjárnánk és megmérnénk a megtett utat. A matematikai modell segítségével a bemenő adatokból kiszámítjuk az úthosszakat. Ilyenkor a modellen tesszük meg a „kirándulást", nem a valóságban. Kísérletet végzünk a modellen, hogy megfigyelhessük az eredményt, azaz az útvonalhosszakat. Ezt nevezzük szimulálásnak.

Rendszerek, jelenségek szimulálására a számítógépet igen széles körben alkalmazzák. A holdraszállás számítógépes lejátszása, kikísérletezése éppúgy szimuláció, mint a játékok.

● v kezdősebességgel függőlegesen feldobunk egy testet. Mennyi idő múlva lesz h magasságban?

A gravitációs erőtér hatására a test egyenletesen változó mozgást végez. Az idő függvényében a test pillanatnyi helyzetét a

$$h = -\frac{g}{2} t^2 + vt$$

képlet segítségével határozhatjuk meg.

Ez lesz a függőleges hajítás matematikai modellje homogén gravitációs térben, és ha nem vesszük figyelembe a közegellenállást.

Akkor, amikor különböző magasság (h) és kezdősebesség (v) értékekhez meghatározzuk – a modell alapján – az időértékeket, tulajdonképpen egy-egy hajítást hajtunk végre. Kísérleteket végzünk a modellel. Lejátsszuk a jelenséget, azaz szimuláljuk azt!

```
10  REM* * *FUGGOLEGES HAJITAS-1 * * *
20  CLS: INPUT "KEZDOSEBESSEG(M/S)",V
30  INPUT "MAGASSAG(M)",H
40  A=-9.81/2:B=V:C=-H
50  D=B↑2-4 * A * C
60  IF D<0 THEN PRINT "NEM ERI EL EZT A
        MAGASSAGOT":END
70  U=-B/2/A:V=SQR(D)/2/A
80  PRINT "A TEST",H;"M MAGASSAGBAN"
90  PRINT U+V,"ES",U-V,"SEC MULVA LESZ"
```

Szimulációs programunkkal egy másodfokú egyenletet oldunk meg és az eredményeket, a feladatnak megfelelően értelmezzük.

A függőleges hajítást írja le a következő algoritmus is:

$S \leftarrow v$

$v_k \leftarrow V$

$a \leftarrow -g$

$V_v \leftarrow V_k + a\Delta t$

$$\Delta S \leftarrow \frac{v_k + V_v}{2} \cdot \Delta t$$

$S \leftarrow S + \Delta S$

$t \leftarrow t + \Delta t$

$V_k \leftarrow V_v$

A test helyzetét Δt időközönként meghatározzuk. Ezalatt az idő alatt a test sebessége v_k-ról v_v-re változik, ha a gyorsulása a

$v_v = v_k + a\Delta t$.

A Δt idő alatt megtett út:

$$\Delta S = \frac{v_k + v_v}{2} \Delta t.$$

A Δt idő alatt megtett utak összege adja a test helyzetét:

$$S \leftarrow S + \Delta S.$$

A nyíllal az értékadást jeleztük: S értéke legyen ΔS-sel nagyobb.

A ciklus elejére való visszatérés előtt gondoskodnunk kell arról, hogy a kezdősebesség értéke az legyen, ami előzőleg a végsebesség volt.

$$v_k \leftarrow v_v.$$

```
 10  REM * * * FUGGOLEGES HAJITAS -2 * * *
 20  CLS: INPUT "KEZDOSEBESSEG(M/S)",V
 30  INPUT "MAGASSAG(M)",H
 40  INPUT "IDOKOZ(S)",DT
 50  CLS: S=0:VK=V:A=-9.81
 60  VV=VK+A*DT
 70  DS=(VK+VV)/2*DT
 80  S=S+DS:T=T+DT:VK=VV
 90  IF(H-S)*(H-S+DS)<=0 THEN PRINT
     "ATHALADT",T-DT,T,"SEC KOZOTT"
100  IF S>0 THEN 60
```

A test Δt idő alatt ΔS utat tesz meg. Eközben áthaladhat h magasságon. Ezt vizsgáljuk a ΔS végpontjainak ismeretében a 90-es sorban.

● Rajzoltassuk ki az előző programmal a függőlegesen feldobott test hely–idő grafikonját!

Mivel a test helyzetét a program Δt időközönként kiszámítja ezért csak a megjelenítéshez szükséges utasításokkal kell kiegészítenünk a programot.

```
 90  IF S<0 THEN 110
100  L=L+1:IF S<47 THEN SET(L,47-S)
110  IF L<127 THEN 60
120  IF INKEY$="" THEN 120 ELSE CLS
```

Rajzoltassuk ki a test pályáját!

A függőlegesen feldobott test egy egyenes mentén mozog, ezért csak akkor tudjuk követni a test mozgását, ha csak egy pont látható a képernyőn.

```
 85  IF H<47 THEN SET(0,47-H)
 90  IF S<0 THEN 130
100  IF S<47 THEN SET(0,47-S)
110  IF S-DS<47 THEN RESET(0,47-S+DS)
120  GOTO 60
130  IF INKEY$="" THEN 130 ELSE CLS
```

● Készítsünk szimulációs programot a függőlegesen felfelé hajított test mozgásának vizsgálatához a közegellenállás figyelembevételével!

A közegellenállásból származó erő a sebesség négyzetével arányos, és a sebességgel ellentétes irányú. A test mozgásegyenlete tehát:

$$\vec{ma} = -\vec{mg} + k \cdot |\vec{v}| \vec{v}$$

amiből a test gyorsulása:

$$\vec{a} = -\vec{g} + \frac{k}{m} |\vec{v}| \vec{v}.$$

Ezt az összefüggést felhasználva, a test pillanatnyi helyzetét meghatározó algoritmusunk:

$t \leftarrow 0$

$v_k \leftarrow v$

$S \leftarrow 0$

$a \leftarrow -g \frac{k}{m} |v_k| v_k$

$v_v \leftarrow v_k + a \cdot \Delta t$

$\Delta S \quad \frac{v_k + v_v}{2} \cdot \Delta t$

$S \leftarrow S + \Delta S$

$v_k \leftarrow v_v$.

$t \leftarrow t + \Delta t$

Az előző program és az algoritmus ismeretében a program megírása nem jelenthet gondot.

● Egy egér össze-vissza szaladgál egy labirintusban. Írjunk programot, amivel az egér mozgását szimuláljuk!

A programmal be lehet mutatni, hogy az „egér", még ha sok lépésben is, de végül megtalálja a kijáratot.

A képernyő egy pontjának véletlenszerű mozgását, amivel az „egér" mozgását szimuláljuk, úgy valósítjuk meg, hogy mindig más szomszédos pont vizsgálatával kezdjük az útkeresést (a 130-as sortól a 180-as sorig).

```
10  REM * * * BOLYONGAS A LABIRINTUSBAN * * *
20  CLS:FOR I=10 TO 20
30  SET(I,10):SET(10,I):SET(I,20):SET(20,I)
40  NEXT
50  FOR I=1 TO 9
60  READ X$
70  FOR J=1 TO LEN(X$)
80  SET(10+VAL(MID$(X$,J,1)),I+10)
```

```
 90  NEXT:NEXT
100  RESET(10,11)
110  X=12:Y=15
120  SET(X,Y):RESET(X1,Y1)
130  ON RND(4)GOTO 140,150,160,170
140  IF POINT (X-1,Y)=0 THEN X=X-1:GOTO 190
150  IF POINT (X,Y-1)=0 THEN Y=Y-1:GOTO 190
160  IF POINT (X+1,Y)=0 THEN X=X+1:GOTO 190
170  IF POINT (X,Y+1)=0 THEN Y=Y+1:GOTO 190
180  GOTO 140
190  RESET(X1,Y1):SET(X,Y):X1=X:Y1=Y
200  IF X<>10 THEN 120
210  DATA 4,24578,27,4578,12,4568,248,2468,6
```

● Ne legyen a labirintusnak olyan elválasztható fala, amelyiket körbe lehet járni. Ilyen labirintusból úgy tudunk kijutni, hogy az egyik kezünkkel (legyen ez a bal kezünk), mindig érintjük a labirintus falát, miközben elõre haladunk.
„Tanítsuk meg" erre az „egeret"!

A programmal azt az algoritmust kell megfogalmaznunk, ami az egér, azaz a képernyõn megjelenõ pont mozgását vezérli.

Tekinthetjük az egeret egy automatának, aminek a mozgását egy programmal irányítjuk.

A szomszédos pontok vizsgálatát mindig a mozgás irányához képest balra elhelyezkedő ponttal kezdjük. K változó értéke jelzi, hogy milyen irányba indult el az „egér" K értékei: balra k = 0, fel k = 1, jobbra k = 2, le k = 3.

K értéke jelzi, hogy hol kezdjük a vizsgálatot (180-as sor). A megfelelő vizsgálat kiválasztása a 110-es sorban történik, és az óramutató járásával megegyező irányban folytatódik, ha az adott irányban fal van, azaz a képernyõn a grafikus pont világít.

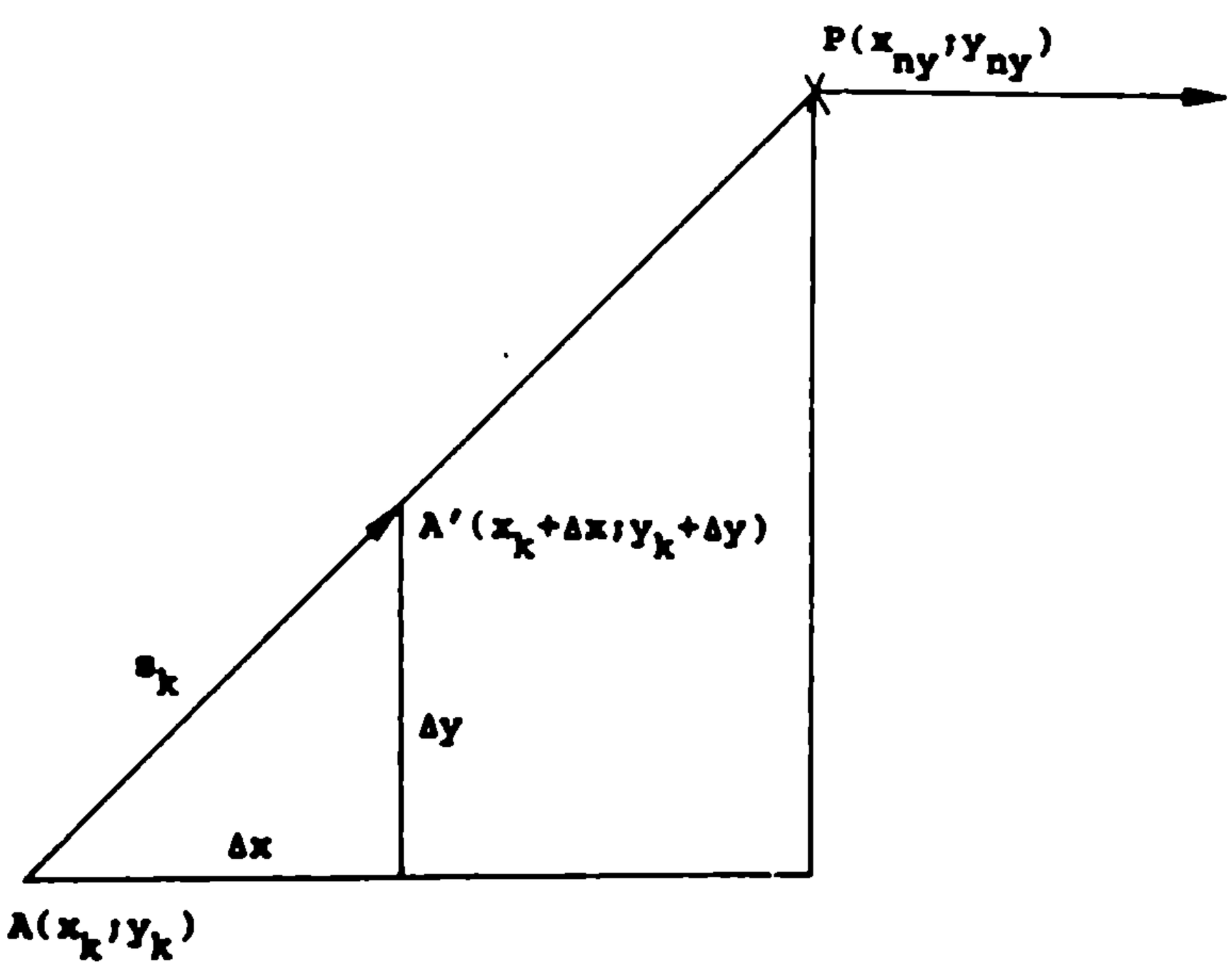
```
 10  REM* * *KIUT A LABIRINTUSBOL * * *
 20  CLS:PRINT@ 2*64+7," ":FOR I=1 TO 5:PRINT
     CHR$(176),:NEXT
 30  PRINT CHR$(13),:FOR I=1 TO 3:PRINT
     STRING$(8,32),CHR$(149),CHR$(192+3),
     CHR$(170):NEXT
 40  PRINT TAB(8),:FOR I=1 TO 5:PRINT CHR$(131),
     :NEXT
 50  FOR J=1 TO 9
 60  READ X$
 70  FOR I=1 TO LEN(X$)
 80  IF MID$(X$,I,1)="1" THEN
     SET(16+I,8+J)
 90  NEXT:NEXT
100  RESET(16,9):X=23:Y=17:X1=X:Y1=X
110  SET(X,Y):ON K GOTO 130,140,150
```

```
120  IF POINT(X-1,Y)=0 THEN K=0:X=X1-1:
     GOTO 170
130  IF POINT(X,Y-1);0 THEN K=1:Y=Y1-1:
     GOTO 170
140  IF POINT(X+1,Y)=0 THEN K=1:X=X1+1:
     GOTO 170
150  IF POINT(X,Y+1)=0 THEN K=3:Y=Y1+1:
     GOTO 170
160  GOTO 120
170  RESET(X1,Y1):X1=X:Y1=Y
180  IF K=0 THEN K=3 ELSE K=K-1
190  IF X=15 THEN PRINT@13 * 64,:END ELSE 110
200  DATA 01001001,010111,00000101
210  DATA 10110101,1011,0001011
220  DATA 010101,01110101,000101
```

● Egy kutya, amikor meglát egy nyulat, és el akarja fogni, mindig abba az irányba szalad, amerre a nyulat látja. Szimuláljuk a kutya mozgását!

A kutya szabályozza mozgását. Folyamatosan figyel a nyúlra, és ennek megfelelően változtatja a futás irányát.

A programban az irány megállapítását Δt időközönként végezzük. Két iránymeghatározás között a kutya mozgását egyenesvonalú egyenletes mozgásnak tekintjük. A kutya. a 71. ábra szerint, az $A(x_k; y_k)$ pontból megpillantja a nyulat a $P(x_{ny}; y_{ny})$ pontba



71. ábra

Attól függetlenül, hogy korábban milyen irányban futott, ezek után Δt ideig v_k sebességgel a P pont irányába megtesz $s_k = v_k \Delta t$ utat. A Δx és a Δy elmozdulás értékeket, azaz

a $A'(x_k + \Delta x; y_k + \Delta y)$ pont koordinátáit az $A\overset{\frown}{A}A'$ és az ARP háromszögek hasonlóságát felhasználva számíthatjuk ki az A és P pontok koordinátáinak ismeretében.

$$\Delta x = s_k \cdot \frac{x_{ny} - x_k}{d}; \quad \Delta y = s_k \cdot \frac{y_{ny} - y_k}{d},$$

ahol $d = \overline{AP} = \sqrt{(x_{ny} - x_k)^2 + (y_{ny} - y_k)^2}$.

Miközben a kutya az A' pontba ér, a nyúl már a P' pontban lesz, és a kutyának ismét módosítania kell a futás irányát.

A kutya akkor fogja el a nyulat, amikor $d < s_k$ egyenlőtlenség teljesül, azaz a nyúl közelebb van, mint egy kutyaugrás.

```
 10  REM * * * KUTYA-NYUL * * *
 20  CLS:INPUT "KUTYA SEBESSEGE",VK
 30  INPUT "NYUL SEBESSEGE",VN
 40  INPUT "IDOKOZ",T
 50  CLS:XN=RND(40):YN=RND(20)
 60  XK=20+RND(50):YK=47
 70  D=SQR((XN-XK)↑2+(YN-YK)↑2)
 80  XN=XN+VN * T
 90  SK=VK * T
100  XK=XK+SK * (XN-XK)/D
110  YK=YK+SK * (YN-YK)/D
120  IF D<SK OR XN>126 OR XK>126 OR YK<0
     THEN 150
130  SET(XN+.5,YN):SET(XK+.5,YK+.5)
140  GOTO 70
150  IF INKEY$="" THEN 150 ELSE CLS
```

Programok címlistája

Billentyűzet használata 44
Életkor 47
Téglatest 49
Gömb 50
Hatványozás 51
Körhenger-1 52
Körhenger-2 52
Értékadás 53
Mértékegységek 53
Pontosság 54
Byte decimális értéke 54
Maradékos osztás 55
Osztályzatok-1 61
Osztályzatok-2 62
Csillagok 64
Karakter mozgatása 64
Képújság 65
Összeg, szorzat 68
Fej vagy írás 69
Elsőfokú egyenletrendszer 73
Hét napjai 74
Kikeresés data-ból 74
Rajtszámok 75
Nyugdíjjárulék 76
Számjegyek betűvel 77
Betűk-1 78
Betűk-2 78
Legkisebb-1 80
Legkisebb-2 80
Legkisebb-3 81
Sorozat elemeinek összege-1 82
Sorozat elemeinek összege-2 83
Sorozat elemeinek összege-3 84
Kikereséses összegzés 86
Négyzetgyök közelítés alulról 89
Legnagyobb közös osztó 90

Háromszög kerulete és területe 93
Newton-féle iteráció 95
Maradékos osztás 96
Kerekítés 96
Printusing 97
Lottó számok 97
Imbolygó karakter 97
Pont-egyenes 1 98
Pont-egyenes 2 99
Szám fordítva 99
Háromszög szögei 100
Szög szinusza 101
Bolygópont-1 103
Bolygópont-2 104
Pattogó golyó 105
Betűk-1 106
Betűk-3 107
Betűk-5 107
Függvény-1 108
Grafikus karakter 110
Függvény-2 110
Konvertálás nyolcasszámrendszerbe 111
Legkisebb-legnagyobb 112
Sorbarendezés 115
Kockadobás 115
Számok kiíratása betűvel 116
Munka kiértékelése 117
Adatok mágnesszalagra 117
Ellenőrzés 118
Teljesítmény a hét napjain 118
Hatványok 119
Karakterláncok-1 122
Karakterláncok-2 122
Konvertálás kettes számrendszerbe 123
Életkor 124
Betűk előfordulása 126
Karakterkészlet 126
Láthatatlan beírás 127
Grafikus karakterek 128
Világos képernyő 128
Téglalap 128
Imbolygó tengerész 129
Átalakítás 129
Függvénytáblázat 130
Lottó szelvények 135
Totó tippek-1 137
Totó tippek-2 137

Dobókocka-1 138
Dobókocka-2 139
Adatrögzítés 142
Rendezés 143
Összefésülés 145
Tulajdonos kikeresése 146
Gyökelkülönítés 148
Gyökfinomítás-felezés 151
Függvény görbéje alatti terület 152
Egyenletrendszer 155
Bástya 157
Kírás-nem ütik egymást 159
Körút 160
Függőleges hajítás-1 161
Függőleges hajítás-2 162
Bolyongás a labirintusban 163
Kiút a labirintusból 164
Kutya-nyúl 166

Tartalomjegyzék

A számítógépek kialakulása, a számítástechnika fejlődése 5
A számítógépek működése és felépítése 7
Az aritmetikai egység 8
A memória 9
A vezérlőegység 10
Az adatok, információk megjelenítése 11
A számrendszerek 12
Aritmetikai műveletek bináris számrendszerben 15
Adathordozók 21
Bevezetés a programozásba 26
A személyi számítógépek kialakulása és működése 33
A programozás különböző szintjei 36
Fortran 38
Algol 39
Cobol 39
PL1 (Programme Language 1) 39
BASIC 39
A BASIC programok kezelése 41
Programozás BASIC nyelven 46
Adatbeolvasás és -kiírás 46
Értékadás 52
A program vezérlése 56
Feltétel nélküli vezérlés 56
Feltételes vezérlő utasítás 56
Vizsgálat 67
Adatok a programban 74
Ciklus 79
Utólagos vizsgálattal szervezett ciklus 63
Előzetes vizsgálattal szervezett ciklus 85
Aritmetikai függvények 93
Grafika 102
Indexes változók 111
Karakterláncok 122
Szubrutin 130
Alkalmazási feladatok 140
Egy összetettebb feladat 140
Numerikus módszerek 147
Kombinatorikus feladatok 156
Szimulálás 160

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Fischer Herbert igazgató

Széchenyi Nyomda, Győr 85. K–433
Felelős nyomdavezető: Nagy Iván igazgató

Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Trencséni Ágnes
A borítót tervezte: Sebes János
A könyv formátuma: B5
Ívterjedelme: 15,375 (A5) + 2,125 (A5) ív melléklet
Ábrák száma: 73
Papír minősége: 80 g-os ofszet
Betűcsalád és méret: Press Roman, 10/12
Azonossági szám: 90052
MŰ: 3767-i-8588
Készült az MSZ 5601 és 5602 szerint